VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

KRYPTOGRAFICKÁ PODPORA SOUČASNÝCH PROGRAMOVATELNÝCH ČIPOVÝCH KARET
CRYPTOGRAPHIC SUPPORT OF CURRENT PROGRAMMABLE SMART CARDS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Bc. Petr Vančo**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Petr Dzurenda**
SUPERVISOR

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Informační bezpečnost**
Ústav telekomunikací

Student: Bc. Petr Vančo ***ID:*** 174958
Ročník: 3 ***Akademický rok:*** 2018/19

NÁZEV TÉMATU:

Kryptografická podpora současných programovatelných čipových karet

POKYNY PRO VYPRACOVÁNÍ:

Student se seznámí se současnými platformami programovatelných čipových karet a s jejich kryptografickou podporou. Výstupem bakalářské práce bude návrh a implementace softwarového nástroje pro testování čipových karet MultOS. Softwarový nástroj bude schopen vypsat informace o kartě, podporované kryptografické a matematické algoritmy, výsledky benchmarkových testů čipových karet apod. Aplikace bude obsahovat také uživatelské rozhraní pro snadnou obsluhu a také bude umožňovat zpracovat získané informace z karty.

DOPORUČENÁ LITERATURA:

[1] RANKL, Wolfgang; EFFING, Wolfgang. Smart card handbook. John Wiley & Sons, 2004.

[2] STINSON, Douglas R. Cryptography: theory and practice. CRC press, 2005.

Termín zadání: 1.2.2019 ***Termín odevzdání:*** 27.5.2019

Vedoucí práce: Ing. Petr Dzurenda
Konzultant:

prof. Ing. Jiří Mišurec, CSc.
předseda oborové rady

ABSTRAKT

Bakalářská práce se zabývá bechmarkovými testy nad čipovými kartami MultOS. V úvodu práce jsou popsány vlastnosti čipových karet, následuje pak komunikace s čipovými kartami. Poté jsou zde srovnány operační systémy čipových karet. Další kapitola se zabývá kryptografickou podporou čipových karet. Je zde také popsáno jaké nástroje se využívají při vývoji aplikací pro operační systém MultOS. V rámci této práce byl implementován systém pro správu a testování čipových karet s operačním systémem MultOS. V poslední části je tedy popsán celý vývoj a to aplikací pro čipovou kartu, tak i aplikace pro PC i s uživatelským rozhraním.

KLÍČOVÁ SLOVA

Čipové karty, MultOS, SmartDeck, benchmark, kryptografie, rychlost kryptografických operací

ABSTRACT

This bachelor thesis focuses on benchmark tests of the MultOS smart cards. The characteristics of the smart cards are described at the beginning of the thesis, next part is about smart card communication. Then there is a comparison of operating systems. Next chapter is about cryptographic support for smart cards. Here is also described, which tools are used to develop for operating system Multos. The system for management and testing smart cards with MultOS was implemented in this thesis. Whole development process is described in last chapter, for smart card application, and for PC application with user interface.

KEYWORDS

Smart cards, MultoOS, SmartDeck, benchmark, cryptography, speed of cryptography operations

VANČO, Petr. *Kryptografická podpora současných programovatelných čipových karet*. Brno, Rok, 48 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: Ing. Petr Dzurenda

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Kryptografická podpora současných programovatelných čipových karet“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu Ing. Petru Dzurendovi za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno ....

podpis autora

Tato práce vznikla jako součást klíčové aktivity KA6 - Individuální výuka a zapojení studentů bakalářských a magisterských studijních programů do výzkumu v rámci projektu OP VVV Vytvoření double-degree doktorského studijního programu Elektronika a informační technologie a vytvoření doktorského studijního programu Informační bezpečnost, reg. č. CZ.02.2.69/0.0/0.0/16_018/0002575.

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Projekt je spolufinancován Evropskou unií.

Obsah

Úvod

Čipovými kartami jsme dnes obklopeni na každém kroku, nejčastěji však v bankovnictví. Setkáme se s nimi i jako s přístupovým klíčem do práce nebo školy. Skoro každý z nás má telefon a v něm SIM (Subscriber Identity Module) kartu, která se také řadí mezi čipové karty. V některých zemích (např. v sousedním Slovensku) se můžeme potkat i s tím, že čip najdeme v pasu nebo v občanském průkazu. Jelikož jsou někdy celé systémy založeny na čipových kartách, je potřeba zaručit jejich bezpečnost. Čím častěji se karty používají, tím větší důraz bychom měli dávat na jejich zabezpečení, délku klíčů a nové kryptografické protokoly. Při výběru karty pro vyvíjený systém může nastat problém pří výběru správné karty. Každý výrobce vyrábí jinak výkonné karty a podporuje jiné funkce, které pro vývojáře mohou být důležité. Vývojář však ve většině případů neví, jaké funkce jsou podporované a když tak s jakými délkami klíčů.

Cílem této diplomové práce je analyzovat vlastnosti a schopnosti současných programovatelných čipových karet. Důraz je pak zaměřen zejména na platformu čipových karet MultOS. Výstupem práce je pak návrh a implementace vlastního softwarového řešení pro testování čipových karet. Díky této aplikaci mohou vývojáři a bezpečnostní experti zvolit implementaci čipové karty, která splňuje bezpečnostní a výkonové požadavky jejich systému.

Úvod práce popisuje rozdělení čipových karet, jednotlivých typů, popis komunikace s terminálem a popis karet. Zároveň jsou zde popsány nejznámější operační systémy čipových karet.

Druhá část práce popisuje aktuální stav čipových karet a jejich kryptografické možnosti a rozdíly mezi systémy. Následně je zde popsáno, jakým způsobem byly získány klíče pro eliptické křivky a pro RSA.

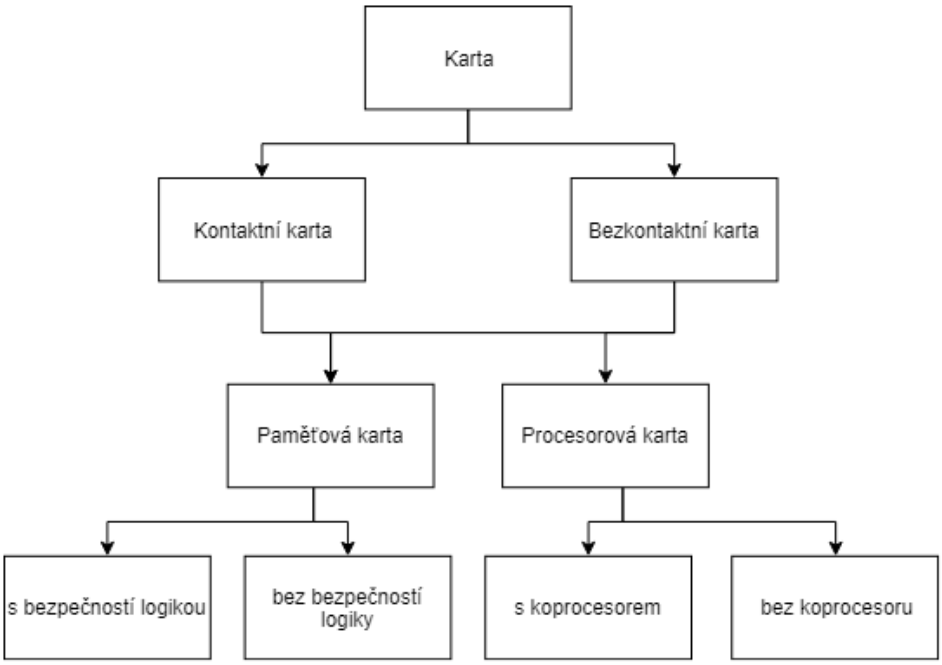
V poslední kapitole je popsána výsledná aplikace, její grafické prostředí, samotné měření, jakým způsobem bylo měření prováděno a zároveň zde naleznete interpretaci naměřených výsledků.

1 Čipové karty

1.1 Vlastnosti čipových karet

Čipová karta je plastová karta, ve které se nachází čip. Většinou je do plastové karty vyfrézován prostor o velikosti čipu, ve kterém je následně čip vlepen. Pokud se jedná o bezkontaktní čip, lze zde nalézt ještě anténu, která je zde také vlepena.

V dnešní době se objevují dva druhy čipových karet a to konkrétně karty kontaktní a bezkontaktní. Vždy se jedná o plastovou kartu, která má definovanou velikost podle normy ISO/IEC 7810 [3]. Čipové karty lze také dělit na paměťové karty a procesorové karty. Nejznámější paměťovou kartou je telefonní karta a procesorovou kartou je platební karta. Vývojář má možnost si vybrat z velkého množství druhů karet, jak je možno vidět na obr. 1.1

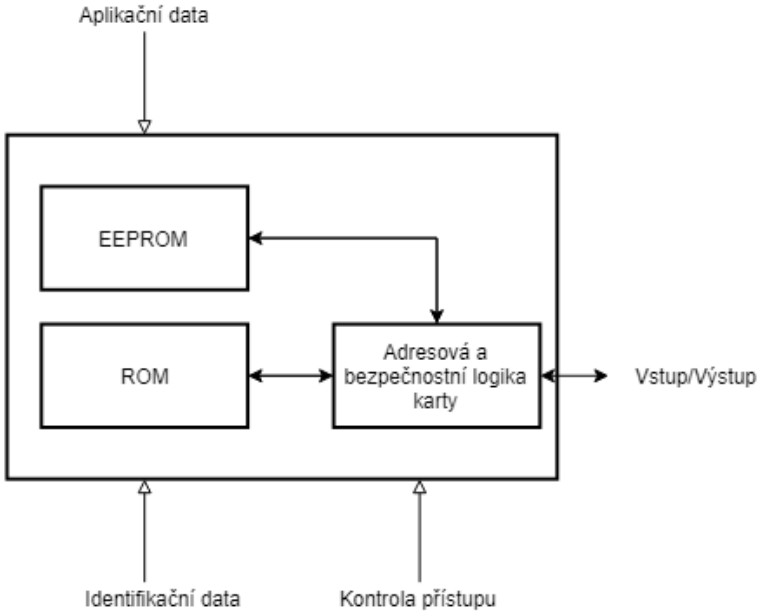


Obr. 1.1: Klasifikace karet.

1.1.1 Paměťové karty

Obsahují na svém čipu pouze elektricky programovatelnou polovodičovou paměť EEPROM (Electrically Erasable Programmable Read-Only Memory) nebo EPROM (Erasable Programmable Read-Only Memory).[2] Blokové schéma paměťové karty je zobrazeno na 1.2. První paměťové karty byly použity pro telefonní budky, kdy na nich byl nahrán předplacený kredit. Tato informace o výši kreditu byla elektronicky

uložena na čipu. Jakmile ji zákazník používal k telefonování, hodnota se snižovala. Používala se tedy pouze paměť a žádný procesor nebyl přítomen. Muselo být však zabezpečeno, aby si uživatel nijak nemohl výši kreditu samovolně zvednout. Kdokoliv by si tak mohl kartu zkopírovat po koupi a po spotřebování kreditu zase nahrát původní data a takto používat kartu do nekonečna. Vnitřní bezpečnostní logika karty zaručila, že nebylo možné mazat určité paměťové buňky, jakmile byly jednou zapsané. Když byla karta bez kreditu nemohla se znovu použít, protože paměťové buňky jsou nepřepisovatelné. I přesto, že čip byl velmi malý a levný, byla škoda jej nepoužít znovu. Tyto prázdné karty se skartovaly. Karty tohoto typu se využívají pro parkovací automaty nebo veřejnou dopravu. Lze říct, že tyto karty jsou vhodné jako předplacené karty a nebo jako jednoúčelové identifikační karty pro systémy, kde nízká cena je nejdůležitější aspekt celého systému. [1]

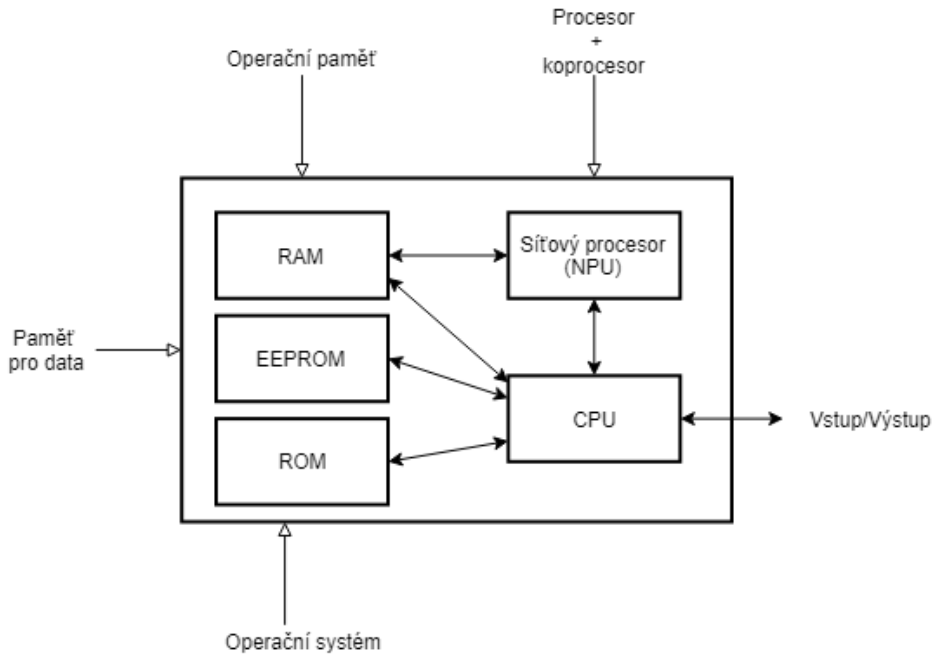


Obr. 1.2: Blokové schéma paměťové karty

1.1.2 Procesorové karty

Na jejich čipu můžeme nalézt mikroprocesor, obvykle 8-bit, 16-bit nebo 32-bitů. Mikroprocesor je schopen provádět různě náročné výpočty. Nejdůležitější operací je šifrování a dešifrování různých dat pomocí klíče, který je na kartě uložen. Díky tomu lze karty využívat např. pro autentizaci. Možnost ukládat šifrovací klíč na kartě a spouštět šifrovací algoritmy dává kartě velké možnosti. Vývojář díky tomu může implementovat např. bezpečné platební systémy. Jakmile jsou na kartě paměť

i procesor, které jsou programovatelné, tak pak využití závisí jen na velikosti výpočetního výkonu, velikosti paměti a zároveň na představivosti vývojáře, jakou aplikaci dokáže vytvořit. Procesor může být také doplněn o koprocesor, který má na starost složité kryptografické výpočty jako je třeba šifrování RSA, AES nebo kryptografické algoritmy založené na principu eliptických křivek např ECDH (Elliptic-Curve Diffie–Hellman), ECDSA (Elliptic Curve Digital Signature Algorithm). Někdy také označován jako kryptoprocesor. Cena takovýchto karet stále pomalu klesá a to díky masové výrobě, aktuálně se cena pohybuje mezi dvěma až deseti americkými dolary za kus. Čipové karty se využívají v mobilních telefonech a bankovních aplikacích, jako identifikační karty do různých objektů, k elektronickým podpisům a nebo pro ukládání citlivých dat. Na některých kartách může být nainstalováno více aplikací naráz s tím, že každá aplikace má své přiřazené místo, kde může ukládat a nesmí přepisovat data jiným aplikacím. Největší výhodou těchto karet je možnost provádět náročné výpočty a ukládat množství dat vzhledem k relativně levné pořizovací ceně. [1]



Obr. 1.3: Blokové schéma procesorové(tzv. čipové) karty

Na obr. 1.3 jsou zobrazeny základní komponenty čipové karty:

- **RAM** - jedná se o operační paměť procesoru
- **EEPROM** - jedná se operační paměť, sloužící pro ukládání dat jednotlivých aplikací na kartě. Přístup do paměti je kontrolovaný bezpečnostní logikou
- **ROM** - jedná se o paměť, která již není editovatelná a je zde uložen operační

systém karty
- **CPU** - výpočetní jednotka karty - procesor
- **NPU** - koprocesor - jedná se o pomocný procesor k CPU, který je zaměřen především na složité výpočty

Paměti

Paměti slouží na uchovaní kódu programu a dat. Paměti jsou RAM (Random-Access-Memory), ROM (Read-Only-Memory) a EEPROM. Velikost pamětí se u různých výrobců velmi liší a to podle toho pro jaký typ použití je karta vytvořena. V novějších modelech se EEPROM nahrazuje flash pamětí, která nezabírá tolik místa na čipu a zároveň je rychlejší. RAM paměť je přepisovatelná, zároveň však pro udržení informací potřebuje elektrický proud. Po odpojení karty dojde ke ztrátě uložených dat. Slouží především pro ukládání dat za běhu aplikace(tzv. session data). EEPROM je přepisovatelná paměť, její výhodou je, že nepotřebuje elektrický proud pro udržení informací. Slouží tedy pro všechny informace, které jsou přepisovatelné a dlouhodobě udržitelné. ROM paměť slouží pouze pro čtení. ROM paměť je napěťově nezávislá. Nachází se zde operační systém karty a sada diagnostických nástrojů, které jsou do paměti vypáleny již při její výrobě. [1]

Procesory

V paměťových kartách se využívají procesory, které jsou vytvořené pouze pro jeden účel. I starší typy procesorů mají výkon stále dostačující. Základní procesory jsou 8-bitové. V těchto procesorech se nachází instrukční sada, jenž je většinou založena na architektuře Motorola 6805. Tyto procesory mohou obsahovat instrukce navíc, například pro adresování 16-bitové paměti, případně obsahují řešení pro překročení 64kB paměti. 16-bitový procesor využívá instrukční sady RISC (Reduced Instruction Set Computer). Dlouho však existoval pouze jediný procesor s touto sadou a to H8 vyráběný firmou Renesas [1]. V dnešní době se již využívají 32-bitové procesory, které poskytují dostatečně velké adresování paměti a dostatečný výkon pro naše aplikace. Největší kritéria při výběru procesoru jsou odolnost proti útoku, spotřeba a hustota kódu. [1]

1.1.3 Rozdělení čipových karet dle způsobu přenosu informací

Způsob komunikace čipové karty se čtecím zařízením můžeme rozdělit na dvě kategorie, a to kontaktní a bezkontaktní. Kontaktní karty komunikují se čtečkou pomocí pozlacených kontaktů.

Bezkontaktní karty žádné pozlacené kontakty nemají. Karta je napájena pomocí elektromagnetické indukce. Karta obsahuje anténu, díky které karta může komunikovat až na vzdálenost 10 cm.

Kontaktní

Na kontaktní kartě jsou umístěny kontaktní plíšky, které poskytují osm konektorů, přes které prochází elektrický proud. Kontaktní oblast má velikost okolo 1cm^2. Dva konektory se využívají pro vstup a výstup, jedná se o střídavou obousměrnou komunikaci. V jednom čase tedy komunikace může probíhat pouze jedním směrem, směr přenosu se však může měnit. Rychlost komunikace se pohybuje okolo 115kbps. Ukázka kontaktu společně s popisem jednotlivých částí je zobrazena na obr. 1.4 [5] [1]

Obr. 1.4: Schéma vývodů čipu podle ISO 7816-2.

- **Vcc** - přívod pro napájení celé karty
- **RST** - pin pro reset
- **CLK** - hodinový signál mezi klientem a serverem, který generuje čtečka
- **GND** - uzemnění
- **Vpp** - tento konektor se již v dnešní době nevyužívá, dříve se však využíval pro programové napájení pro EEPROM paměti.
- **I/O** - vstup/výstup obousměrná komunikace mezi klientem a serverem

C4 a C8 jsou rezervované pro budoucí použití. Konkrétní rozmístění kontaktů na plošném spoji je zobrazeno na 1.4 Pro tuto část byly čerpány informace z [8, 9]

Bezkontaktní

Bezkontaktní karty dokáží komunikovat mezi klientem a serverem bez použití reálného kontaktu. Karta musí obsahovat vestavěnou anténu v plastovém těle. Nejsou vyžadovány žádné viditelné kontakty. Jakmile je tato karta přiložena dostatečně blízko ke čtečce, která disponuje další anténou, může procházet energie ve formě elektromagnetických vln mezi kartou a čtečkou. Vlny jsou vysílány ze čtečky a přijímány na kartě. Jedná se o velmi malý proud, který je však dostatečný pro napájeni

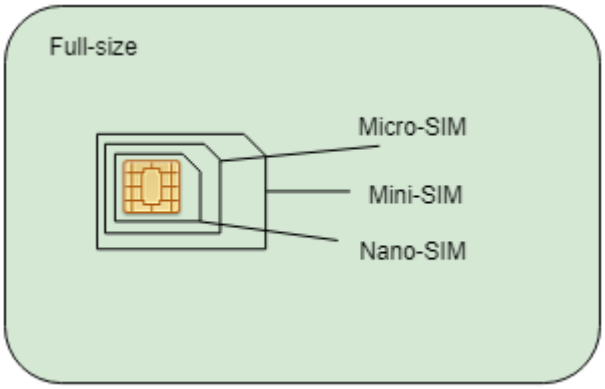
karty. Karty fungují na frekvenci 13.56 MHZ na vzdálenost do 10cm a přibližná rychlost přenosu dat je 100Kbps. [5] [6]

Tento typ karet je v dnešní době velmi rozšířený díky jednoduchosti použití a bezpečnosti. Bezkontaktní karty mají větší životnost a spolehlivost než karty kontaktní. Má to i své negativa, a to je bezpečnost. Jelikož přenos je přes rádiové vlny, je možné, že někdo naši komunikaci zachytí jeho odposlouchávacím zařízením. Dalším nebezpečím je náhodná komunikace s terminálem, se kterým uživatel komunikovat nechtěl. Takový terminál může mít útočník kdekoliv schovaný a ten může začít komunikovat s uživatelovou kartou. To má pak například za následek, že uživatel přijde o svoje peníze z bankovního účtu. [5]

Konstrukce karty

Většina dnešních čipových karet je vytvořena z vrstev různých materiálů. Různé materiály dávají kartě různé vlastnosti a různou dobu životnosti. Většinou jsou karty vyrobeny z polyesteru, polycarbonátu nebo PVC materiálu. Všechny vrstvy karty jsou vytištěny zvlášť a následně slisovány velkým tlakem dohromady. Potom je zde vyfrézováno místo pro čip, který je zde poté umístěn. [7]

Každá karta má definovanou velikost podle standardu [3], všechny typy karet však mají společnou tloušťku a to konkrétně 0,76mm. Díky tomu, že kontaktní karty se vkládají do různých čteček, je potřeba mít jasně definované rozměry, aby všechny karty komunikovaly se všemi terminály správně. U většiny platebních terminálů se používá formát ID-1 na obr. 1.5 označena jako Full-size. Například v mobilních telefonech jsou v dnešní době běžné tři velikost karet, což v některých případech činí uživatelům problémy. Různé velikosti karet jsou zobrazeny na obr. 1.5 spolu s popisem v tab. 1.1.



Obr. 1.5: Zobrazení různých velikostí karet.

	Výška	Šířka
Full-size	85.6 mm	53.98 mm
Mini-SIM	25 mm	15 mm
Micro-SIM	15 mm	12 mm
Nano-SIM	12.3 mm	8.8 mm

Tab. 1.1: Detailní popis velikostí čipových karet[10]

1.2 Komunikační rozhraní

1.2.1 Princip komunikace mezi čtečkou a kartou

Hodinový signál

Procesorové karty nemají integrovaný generátor času, proto vnější hodinový signál je nutný pro všechny komunikace, které probíhají mezi čtečkou a kartou. Tento signál je referenční pro přenos dat. Hodinový signál, který je připojen na kontakt karty však nemusí mít stejnou frekvenci jako signál procesoru. Procesor totiž může mít vnitřní násobič případně dělič, který způsobí změnu frekvence signálu.[1]

1.2.2 Transportní protokoly čipové karty

Může se stát, že nastane chyba při přenosu dat. Toto se stane tehdy, když čtečka i karta pošlou data ve stejný čas kvůli špatné komunikaci mezi kartou a terminálem. Nastává problém, protože data se přenášejí přes jeden vodič pomocí C7 - I/O viz. obr. 1.4. Nejen, že takto máme problém na aplikační úrovní, může se také stát, že napětí bude příliš veliké a zničí nám to hardware samotné karty. Aby se tomu zabránilo, je v kartě nainstalována ochrana v podobě tranzistoru, který se rozepne v případě velkého napětí. [1]

Komunikace mezi čipovou kartou a okolním světem je na základě dvou transportních protokolů T=0 a T=1. Některé implementace karet však můžou podporovat pouze jen jeden z transportních módů. Což pak znamená, že některé karty nedokáží komunikovat se všemi terminály. Dříve přenos dat společně s příjmem dat byl kontrolován pouze operačním systémem bez jakékoliv hardwarové podpory. To však způsobovalo možné chyby v programech, největším problémem však bylo to, že kontrola byla příliš pomalá oproti řešení přímo na kartě, které je výrazně rychlejší. Některé karty využívají pro přenos protokol USB (Universal Serial Bus). [1] Datová jednotka se nazývá TPDU (Transpor Protocol Data Unit). T=0 je protokol, který je bajtově orientovaný naproti tomu protokol T=1 je protokol orientovaný blokově. T=1 je mnohem složitější a největší výhodou je, že je zde implementovaná detekce

a znovu odesílání chybových bloků. Tento přenos se využívá například u platebních karet.T=0 můžeme najít například v SIM kartách.

Při prvním vložením karty do čtečky, zašle čtečka příkaz pro reset a karta na tento příkaz odpoví zprávou ATR (Anwer To Reset). ATR zpráva obsahuje informace o přenosových rychlostech, transportních protokolech a informacích o vložené kartě. Poté následuje sekvence zpráv PPS (Protocol Parameter Selection) a to však pouze v případě, že chce změnit parametry, které získal v předešlé ATR zprávě. [1]

Transportní protokol T=0

Tento protokol využívá pro přenos základní jednotku jeden bajt. Jedná se o starší typ protokolu. Je velmi složité tento protokol zařadit do referenčního OSI (Open Systems Interconnection) modelu, z toho důvodu, že je provázán v různých vrstvách tohoto modelu. V tomto módu, zahajuje komunikaci vždy čtečka karet. Funguje zde model klient-server, kdy server je karta a čtečka karet je klient. Čtečka vytváří dotazy tzv.(APDU command) a karta pouze odpovídá tzv. (APDU response). Začátek komunikace vypadá tak, že terminál zašle minimálně čtyři bajty hlavičky příkazu. Tato hlavička jasně definuje, jak by se karta měla zachovat. Detailnější popis APDU (Application Protocol Data Unit) zpráv je uveden v kapitole 1.2.3

Protokol T=0 také definuje způsob detekce a zotavení se z chyb, které mohou nastat v průběhu běhu aplikace. Tato detekce musí být implementovaná jak na kartě tak na čtečce karet. Chyba v tomto případě znamená, že aktuálně načtený znak není správný, tedy je neúplný případně chybí paritní bit. [5]

Transportní protokol T=1

Jedná se o transportní protokol, který je orientován blokově. Pokud potřebujeme zvýšit přenosovou rychlost, je zapotřebí využít tento transportní protokol. Jedná se pouze o malé změny oproti T=0. Blok může nést řídící data protokolu případně aplikační data. Protokol jasně definuje řízení toku dat a zotavení se z chyby. Tento protokol se nevyužívá pouze mezi čtečkou a kartou, ale také mezi terminály a terminály s počítači proto, aby si mohla vyměňovat kontrolní data. [1]

1.2.3 APDU zprávy

APDU (Application Protocol Data Unit) je komunikační protokol mezi čtečkou karet a čipovou kartou. Tato zpráva je definovaná standardem ISO/IEC 7816-4[4]. Máme dva typy této zprávy, záleží na tom kdo zprávu odesílá. Pokud tuto zprávu odesílá terminál, nazýváme ji APDU příkaz (APDU command), pokud karta tak pak ji

nazýváme APDU response. APDU zprávy jsou využívány pro výměnu všech dat mezi terminálem a kartou.

APDU command

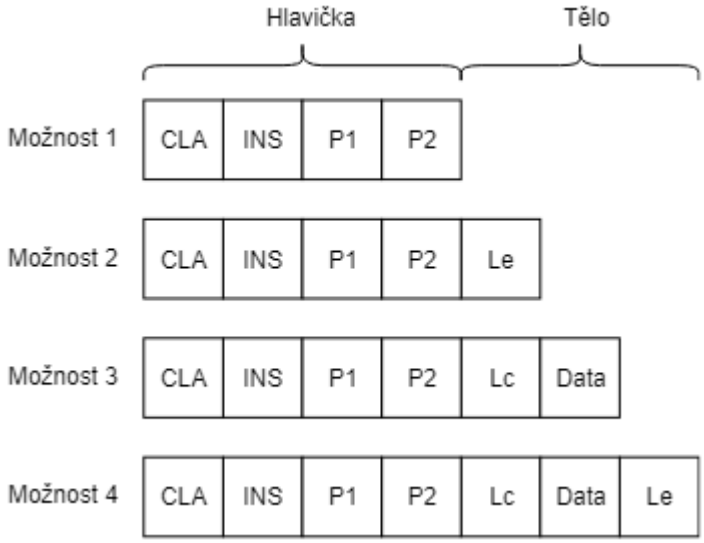
APDU příkaz pro T=0 může mít maximální velikost 4+2+255=261 bajtů. Jedná se o 4 bajty, které jsou povinné, dva volitelné a maximálně 255 bajtů uživatelských dat v jedné zprávě. APDU command můžeme rozdělit na hlavičku a tělo. Do hlavičky můžeme zařadit:

- **CLA** - instrukční třída slouží k identifikaci instrukční třídy, určuje typ příkazu
- **INS** - jedná se o instrukční kód pro výběr aplikace
- **P1,P2** - parametry zprávy, slouží například pro offset v souboru pro zápis dat

Hned za hlavičkou následuje tělo a to konkrétně:

- **lenght command (Lc)** velikost dat v bajtech , které přenáším v APDU zprávě
- Data o maximální velikosti 255 bajtů
- **Le Field (Le)** -velikost zprávy v bajtech, kterou očekávám, že dostanu od karty, toto pole je nepovinné

APDU zpráva může mít čtyři různé podoby, které jsou zobrazeny na obr. 1.6



Obr. 1.6: Čtyři možnosti zprávy APDU

- Možnost 1: definuje scénář, kdy se na kartu posílá pouze hlavička bez dat s tím, že se neočekávají žádná data od karty.
- Možnost 2: posílá to stejné jako v případě prvním, obohacené navíc o jeden byte, který informuje, jak velká data jsou očekávána.

- Možnost 3: posílají nějaké data kartě, ale neočekává se, že nám karta bude data posílat zpět.
- Možnost 4: posílají data a zároveň program očekává odpověď v podobě dat od karty.

Možnost 4: se posílají data a zároveň program očekává odpověď v podobě dat od karty.

APDU response

APDU response je vždy posílána směrem na čtečku. Odpověď se skládá z volitelného těla a povinného zápatí. Zápatí má velikost dva bajty. Jedná se o:

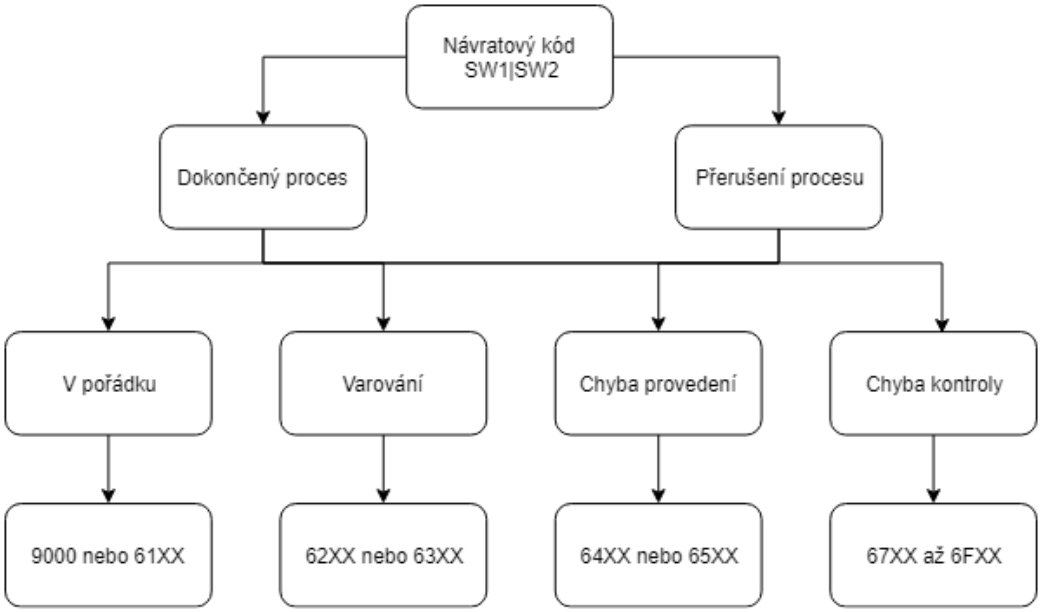
- Velikost dat - proměnlivá, maximální velikost však je definována v položce Le v APDU příkazu. Největší množství přenesených dat v jedné zprávě je 65535 bajtů.
- SW1 - (status word 1) jedná se bajt, který znázorňuje o jakou chybu (např '0x67') se jedná, případně indikuje úspěch operace
- SW2 - (status word 2) jedná se bajt, který specifikuje detaily chyby (např '0x10')

Tyto dva byte SW1 a SW2 se nazývají vratné kódy, které obsahují odpověď na APDU command. Operační systém obsahuje více než 50 různých návratových kódů, které může karta vrátit. Schéma kódů je zobrazeno na obr. 1.7. Návratový kód '0x9000' znamená, že program byl vykonán úspěšně a bez chyb. Kód '0x61XX' je úspěšný s tím, že 'XX' charakterizuje počet dat, které jsou stále dostupná a je možné s nimi pracovat při dalším příkazu. Kód '0x62XX' a '0x64XX' značí, že stav napěťově nezávislé paměti se nezměnil. Návratový kód začínající '0x63XX' a '0x65XX' znamenají, že data v EEPROM nebo ve flash paměti byly pozměněny, ostatní '0x6X' kódy toto neindikují. Pokud se někdy v době běhu programu vyskytne chyba, Le může být nula. [1]

1.3 Operační systémy čipových karet

Mezi nejznámější a nejvíce používané operační systémy jsou zařazeny Java Card, MultOS, Basic Card a .NET Card. Co se týče možností operačních systémů, jsou dosti různorodé. Při zvolení správného operačního systému, je možné využít implementované funkce pro vyvíjenou aplikaci. Vlastnosti karty tedy záleží především na softwarových vlastnostech, než na těch hardwarových, kde se karty dost podobají.

Většina systémů je založena na známých programovacích jazycích jako je Java, jazyk C či C#. Díky tomu mohou vývojáři bez problému začít rychle vyvíjet jejich vlastní aplikace. Vývojář však musí vyvíjet dvě aplikace naráz, a to konkrétně



Obr. 1.7: Návratové kódy definované standardem ISO/IEC 7816-4

aplikaci pro terminál, tak i aplikaci pro kartu samotnou.

1.3.1 Java Card

Java Card je technologie, která dovoluje jednotlivým aplikacím (někdy označovány jako Java Card applet) aby mohly běžet bezpečně na kartě. První Java karty se objevily v roce 1996, kdy cílem výrobců bylo především využít programovacího jazyka Java i v čipových kartách. Mnoho karet založených na Jave, je používáno jako SIM karty, kreditní karty, ID karty případně e-Pasy[13] Aplikace na kartě je stavový konečný automat, který provádí příchozí příkazy, poté může odpovědět zpět buď stavem případně daty. Java Card dovoluje, aby bylo na kartě nahráno více aplikací, které jsou bezpečně od sebe oddělené tj. že jedna aplikace nemůže číst data druhé.

Při vývoji těchto karet se muselo upravit několik věcí oproti Jave na desktopové platformě. A to hlavně JVM (Java Virtual Machine) na JCVM (Java Card Virtual Machine), který nám zajišťuje běh aplikace. Dále se muselo vyvinout nějaké prostředí a funkce, které jsou již dnes definované jako Java Card API. Následně muselo být upraveno JCRE (Java Card Run-time Environment). JCRE vykonává takzvaný bajtkód, nad kterým provádí verifikaci při nahrávání samotné aplikace. Aplikace se tedy před nahráním na samotnou kartu spustí a ověří. Takto ověřená aplikace je poté podepsána a nainstalována na čipovou kartu. Karta samotná má také ověření bytekódu, avšak ne tak důkladné. Pokud aplikaci jen trošku pozměníme, musíme

celý tento proces provést znovu.

Systém Java Card byl vyvíjen s úmyslem především pro uchovávání citlivých informací na kartě. Všechna data ukládají samotné aplikace, které spouští JCVM, který se stará o to, aby jednotlivé aplikace běžely zcela odděleně tak, aby nebylo možné žádným způsobem získat data uložená jinou aplikací. Hardware na kterém systém běží, je vybaven procesorovou akcelerací pro asymetrickou a symetrickou kryptografii včetně podpory operací eliptických křivek. [11] [5]

1.3.2 MultOS

MultOS je multi-aplikační operační systém pro čipové karty, který dovoluje nahrát více aplikací na jednu kartu. Miliony uživatelů v bankovnictví případně ve státních institucích využívají právě MultOS karty. Můžeme je také nalézt v kartách pro bezkontaktní platby, ID karty, e-Pasy případně přístupové karty.

Operační systém MultOS je nejvíce vyzrálým a bezpečným systémem na dnešním trhu. Prvně tyto karty byly psány čistě v bajtovém asemblerovém kódu. Tento jazyk má název MEL (Multos Executable Language) a je založen na programovacím jazyku Pascal. MEL překladač existuje jak pro jazyk C tak pro jazyk Java. Jádro virtuálního stroje operačního systému MultOS obsahuje 31 instrukcí. Každá implementace karty musí obsahovat tuto základní sadu instrukcí, jenž jsou důležité pro běh všech služeb na operačním systému. Tyto služby mohou být rozšířeny, a často bývají, o pokročilejší sadu jako například podporu pro modulární aritmetiku, případně kryptografické algoritmy. Největší výhodou tohoto sytému je, že můžeme využívat ve vysokoúrovňovém programování volání asemblerových a nízkoúrovňových kódů. Vysokoúrovňový kód je poté přeložen pomocí MEL do bajtového kódu.[5] [12] Aplikace má plný přístup ke svým datům, nemá přístup k datům jiných aplikací. MultOS nepodporuje klasický souborový systém pro aplikace, podporuje však statické bloky v EEPROM paměti, kde si vývojář může vytvářet bloky jak potřebuje. Tato data jsou chráněna před přístupem od jiných aplikací. Aplikace také může využívat napěťově závislé paměti. Dynamická data jsou soukromé pro aplikace, nachází se zde například zásobník. Další částí jsou veřejná data, která obsahují data pro přenos z karty do terminálu a zpět.[5] MultOS karty mají takovou podporu, že různé aplikace můžeme kdykoliv nahrávat, či mazat. To je výhodné pro znovupoužitelnost těchto karet. Toto zajišťuje virtuální stroj. Virtuální stroj nám také kontroluje, zdali naše aplikace nesahají do paměti již nahraných aplikací. Pokud ano, virtuální stroj tyto operace zamítne. Na kartě se také nachází část s názvem STEP (Secure Trusted Environment Provisioning), která se stará o bezpečnost aplikačních dat a kódů aplikací.

1.3.3 Basic Card

Basic Card je čipová karta, která je programovatelná v jazyce ZCMBasic, který vychází z jazyku Basic. Basic byl vytvořen v době, kdy počítače měly ještě velmi omezené možnosti. Takto omezené možnosti však mají dnes čipové karty a proto je ZBasic pro karty vhodný. ZCM obsahuje většinu běžných funkcí z jazyka Basic, jako například až čtyř bajtové integery, řetězce a datový typ pro práci s desetinou čárkou (float). Basic Card je zaměřen na co nejrychlejší vývoj aplikace pro kartu a ISO kompatibilitu. Vývojové prostředí je jednoduché k použití. Operační systém obsahuje předdefinované příkazy pro nahrávání aplikací na kartu, které dokonce umožňuje zapnout automatické šifrování APDU komunikace. Pro Basic Card je implementováno velké množství kryptografických funkcí, včetně algoritmů a operací na eliptické křivce, RSA a SHA-512. Kompilátor ZCMBasic převede zdrojový kód na tzv P-code. P-code je jazyk, který je interpretován virtuálním strojem. [14] [5]

2 Kryptografická podpora čipových karet

2.1 Porovnání kryptografických operací mezi operačními systémy čipových karet

Rychlost kryptografického algoritmu je kritickým parametrem všech čipových karet a má přímý vliv na časovou složitost celého kryptosystému. Zvláště u přístupových systémů je vyžadována odezva systému do cca 1 s, v opačném případě se systém jeví uživateli jako pomalý a uživatelsky nepřívětivý. Při návrhu kryptografického protokolu je tedy nutné zvolit takovou platformu čipových karet, která plně vyhovuje požadavkům systému. Proto je vhodné karty prvně otestovat, aby bylo zjištěno, že daná karta s operačním systémem je vhodná pro danou aplikaci. Všechny dostupné operační systémy podporují nějaké kryptografické operace, kterým vývojáři mohou důvěřovat ohledně bezpečné implementace s dostatečnou délkou klíčů. To vývojáři zaručí, že data, která jsou označována jako citlivá, budou v bezpečí při přenosu, případně při provádění různých operací. Na následující tab.2.1 je zobrazeno srovnání jednotlivých kryptografických operací společně s délkou klíčů, které jsou podporované v určitých operačních systémech. Tabulka je vypracována z dokumentací pro vývojáře od jednotlivých výrobců. To však neznamená, že tyto funkce z tabulky najdeme na každé kartě. Kryptografická podpora konkrétní implementace čipové karty pak závisí na samotném výrobci karty. Jednotliví výrobci čipových karet vyrábí karty pro různé oblasti využití a tedy implementují pouze určitou podmnožinu kryptografických operací definovaných samotným operačním systémem. Z tab. 2.1 je zřejmé, že JavaCard, má velké možnosti v asymetrické kryptografii stejně tak jako Basic Card, co však Java Card postrádá je kompletní podpora pro modulární operace, chybí i podpora základních operací na eliptické křivce, ECDH se zde však nachází. Java Card, jako jediný operační systém, podporuje SHA-3, který patří k nejbezpečnějším z rodiny SHA. Operační systém MultOS podporuje široké spektrum funkcí, modulární operace v plném rozsahu i eliptické křivky, chybí však SHA-3. Podpora RNG/TRNG záleží na konkrétní kartě a nemůže být předem zvolena vývojářem. Basic Card je podobně podporovaný jako MultOS, některé funkce podporuje v delších klíčích. Absence je zde pouze na inverzi na eliptických křivkách. .NET karty jsou již dnes zastaralé, proto nejsou v tabulce uvedeny.

2.2 JCAlgTest

Jedná se o automatizovaný systém, který testuje kryptografickou podporu na čipových kartách s operačním systémem JavaCard. Zjišťuje informace ohledně výkon-

		Java Card	**MultOS**	**Basic Card**
Symetrická kryptografie	AES	256 bitů	256 bitů	256 bitů
	3DES	192 bitů	192bitů	168 bitů
	MAC	ANO	ANO	ANO
Asymetrická kryptografie	ECDH	ANO	ANO	Pouze ANO
	RSA	4096 bitů	2048 bitů	4096 bitů
	DSA	1024 bitů	Pouze na eliptických křivkách	Pouze na eliptických křivkách
	EC	512 bitů	512 bitů	544 bitů
Hashovací funkce	SHA-1	160 bitů	160 bitů	160 bitů
	SHA-2	512 bitů	512 bitů	512 bitů
	SHA-3	512	NE	NE
	MD5	ANO	NE	NE
Generování náhodných čísel		TRNG	RNG/TRNG	TRNG
Modulární aritmetika		ANO	ANO	ANO
Operace na eliptické křivce		F_{2^m} a F_p 193/521 bitů	F_p 512 bitů	F_{2^m} a F_p 211/544 bitů
	Sčítání	NE	ANO	ANO
	Násobení	NE	ANO	ANO
	Inverze	NE	ANO	NE

Tab. 2.1: Podporované operace u různých operačních systémů

nosti a podporovaných algoritmů na kartě v různých nastaveních.Pro spuštění testu je potřebné nahrát předem připravený applet na kartu a pustit test. Karta se během deseti minut otestuje a výstupem je textový soubor (CSV) s podporou všech funkcí. Pokud karta funkci podporuje, je zde změřena i rychlost operace. Desktopová aplikace je založena na programovacím jazyku Java a je tedy multiplatformní. Jedná se o command-line aplikaci bez grafického uživatelského rozhraní. Aplikace se i přesto velmi jednoduše ovládá a je jednoduchá i pro úplné začátečníky. Následně je i zde podpora pro interpretaci výsledků, kde další část tohoto programu vizualizuje naměřené výsledky do grafů a tabulek.[16]

3 Vývoj MultOS aplikací

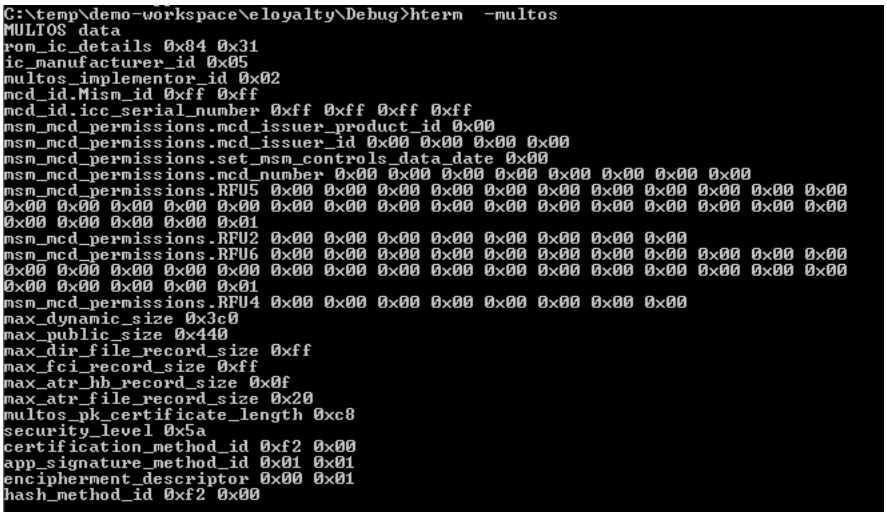
Na základě analýzy současného stavu byla zjištěna potřeba testovacích nástrojů, která existuje pouze pro Java Card. Podobná aplikace, avšak pro čipové karty s operačním systémem MultOS, byla vytvořena v rámci této bakalářské práce, avšak s grafickým rozhraním.

Cílem této části bakalářské práce je vytvořit aplikace jak pro čipové karty, tak pro terminál, které budou kontrolovat kryptografickou podporu jednotlivých operací. Zároveň budou testovat výkon jednotlivých karet měřením několika matematických a kryptografických operací. Aplikace podporuje karty s operačním systémem MultOS. Aplikace má své grafické uživatelské prostředí, které je velmi jednoduché a intuitivní. Výstupy jsou exportovány do souboru (CSV). Aplikace také podporuje správu aplikací na kartě.

3.1 Hardwarové vybavení

Pro tuto bakalářskou práci bylo poskytnuto následující zařízení viz. tab. 3.1. Zde je zobrazena, velikost pamětí společně s frekvencí karet, kde karty ML3-80K-R1 a ML4-P17 jsou vybaveny mnohem větší pamětí, frekvence je však menší, lze tedy očekávat, že ve většině případů pomalejší. Konkrétně se jedná o čtecí zařízení OMNIKEY 3121 připojitelné přes rozhraní USB, dále pak o karty s operačním systémem MultOS ML3-80K-R1 od firmy MultOS a MultOS karta ML421 s operačním systémem MultOS 4.2.1. dále pak karta MultOS karta ML4-P17 s operačním systémem MultOS 4.4.0. Přidělené karty jsou určeny pro vývojáře a pro experimentální testovaní. Karty nevyžadují pro instalaci a odinstalování apletů certifikáty. Kromě certifikátů se karty od běžných karet vůbec nijak neliší, HW a SW podpora je stejná. Certifikáty pro běžné karty se vygenerují online po registraci u společnosti MultOS nebo přímo u výrobce karet.

Základní operace jsou popsány v tab. 3.2. Zde lze vidět, že karta ML3-80K-R1, se zaměřuje na kompletní funkce, chybí jí však základní operace na eliptických křivkách. ML4-P17 je nejméně vybavena ze základními operacemi a není zde implementována žádná podpora výpočtů na eliptických křivkách. Některé operace jsou stejné, avšak jejich implementace se může lišit. Stejný případ s tím, že výrobci nepřidávají implementace funkcí do svých karet, najdeme i u jiných společností vyrábějících čipové karty a to především kvůli hardwarové podpoře.



Obr. 3.1: Výstup z programu hterm, při použití karty ML3-80K-R1

	ML421	**ML3-80K-R1**	**ML4-P17**
ROM	252KB	280KB	neznámá velikost
RAM	1809B	2048B	3500B
EEPROM	18KB	96KB	35KB
Napájení	3-5V	1,62 - 5,5V	1,62 - 5,5V
Frekvence	1 - 10MHz	1 - 7,5MHz	1 - 7,5MHz

Tab. 3.1: Porovnání technických specifikací testovaných karet s operačním systémem MultOS.

3.2 SmartDeck

Pro vývoj těchto aplikací bylo využíváno vývojové prostředí, které bylo připraveno on firmy MultOS. Jedná se o systém SmartDeck, který je poskytován zdarma od výrobce. Tento systém pracuje nad operačním systémem Windows. SmartDeck má v sobě zabudovaný nástroj pro překlad aplikace a také jednoduchý debugger, kterým můžeme simulovat funkčnost aplikace. V tomto balíku aplikací jsou zahrnuty nástroje pro nahrávání programů na čipové karty, případně další podpůrné a komunikační nástroje. Bohužel pomocí debuggeru není možné žádnou aplikaci pro MultOS ladit přímo na kartě, což je obrovský problém při vývoji, jelikož se chování všech karet mírně liší. Zároveň debugger se nechová stejně jako čipové karty, to znamená, že aplikace je potřeba ladit vícekrát. Zabudovaný debugger je však nápomocný, a to hlavně v začátcích vývoje a to pro grafické znázornění jak aplikace funguje. Informace o kartě lze získat zadáním příkazu `hterm -MultOS` v příkazovém řádku. Zde se dozvíte informace o velikosti paměti a úrovni zabezpečení karty. Na obr. 3.1 je

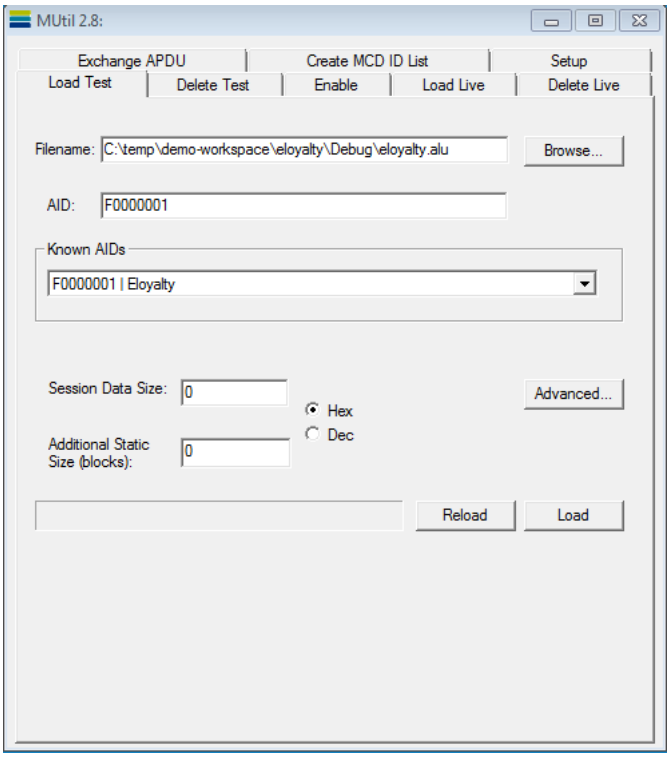
	ML421	ML3-80K-R1	ML4-P17
3DES ECB	NE	ANO	ANO
AES ECB	NE	ANO	NE
ECDH	NE	ANO	NE
ECC Inverze	ANO	NE	NE
ECC Násobení	ANO	NE	NE
ECC Sčítání	ANO	NE	NE
Genrování náhodných čísel	ANO	ANO	ANO
RSA	ANO	ANO	ANO
Modulární inverze	ANO	ANO	NE
Modulární redukce	ANO	ANO	ANO
Modulární násobení	ANO	ANO	ANO

Tab. 3.2: Porovnání dvou různých karet se stejným operačním systémem MultOS.

zobrazen výstup z programu `hterm`, kde vypsán, id výrobce, velikost pamětí, velikost jednoho záznamu, úroveň bezpečnosti, certifikační metoda a další informace o kartě.

3.2.1 MUtil

Výstupem z vývojového prostředí SmartDeck je soubor s příponou .hzx. Tento soubor se musí pomocí aplikace halugen překompilovat do souboru s příponou .alu. Tento soubor už je možné nainstalovat přímo na kartu. Toto nahrání se provádí pomocí nástroje MUtil od firmy MultOS. MUtil je nástroj, který umí pracovat s kartami všech výrobců s operačním systémem MultOS. Před nahráním samotné aplikace, se musí v aplikaci MUtil zvolit AID (Application ID), ten pak dovoluje nahrát na kartu více aplikací, případně přehrát již aktuálně nahranou. Ukázka programu MUtil je zobrazena na obr. 3.2. Při nahrávání se mohou vyskytnou různé potíže, kdy karta vrací různé chybové kódy. Nahrávání není vždy stabilní a může se dostat do problémů. Při pokusu o nahrání více aplikací se karta zablokovala, s tím, že je již nepoužitelná. Bylo změněno AID a byla nahrána aplikace na kartu, další aplikace se tam již nevejde, kvůli nedostatku místa na kartě. Avšak tyto dvě aplikace nejdou ani přehrát a ani dokonce smazat, při pokusu o přehrání karta vrací problém `Error:Delete MEL Application Error 6a81- Unkonwn Error` , při pokusu o nahrání karta vrátí `Error: Openng MEL Application-9d12- Duplicate AID`.



Obr. 3.2: Program MUtil.

3.2.2 Nástroj hterm

Nástroj `hterm` běžící v příkazovém řádku, jako hlavní přednosti lze zařadit snadná komunikace se čtečkou a kartou. Je to nástroj, který je zahrnut ve vývojovém balíčku SmartDeck. Jeho funkce jsou například, zobrazení připojených čteček a jejich stavů, případně dokáže komunikovat přímo s kartou, pracovat s certifikáty a nahrávat aplikace na kartu. Může přímo komunikovat s určitou aplikací pomocí apdu příkazů. Zde je výčet nejpoužívanějších přepínačů programu `hterm`

- `hterm -readers` zobrazí všechny připojené a nainstalované čtečky karet včetně jejich jmen. Pokud je nainstalována pouze jedna čtečka, je vybrána jako výchozí. V případně, že je nalezeno více čteček, je potřeba používat přepínač `-pcsc` pro výběr čtečky.
- `hterm -card` tento příkaz zobrazí typ karty. Typ karty může být HitachiV3, HitachiV4, KeycorpInfineonV4, KeycorpInfineonV4P ,KeycorpInfineonV4I4D, MI-M3, MI-M4. Typ karty se však nemusí zobrazit pokaždé, při testování některých vývojářských karet se zobrazilo, že typ karty je neznámý.
- `hterm -load .hzx` slouží pro nahrání aplikace přímo na kartu. Výhodou tohoto řešení je, že se nemusí využívat nástroj halugen pro překlad zdrojového kódu.

- `hterm -deleteaid aid` při zadání aid, se smaže aplikace na kartě se zadaným aid.
- `hterm -dir` zobrazí všechny aplikace nahrané na kartě. Funguje však pouze pro aplikace nahrané pomocí programu MUtil
- `hterm -apdu` složí k poslání APDU příkazu, vhodné na testování při vývoji

u všech příkazů je možné použít příkaz `-verbose`, který zobrazí více informací o tom, co se zrovna děje a v jakém stavu aktuálně karta je.

Ne všechny příkazy pro aplikaci `hterm` však fungují spolehlivě, ze zkušenosti po použití přepínače `-clean`, program zahlásil úspěšnou operaci, avšak nevymazaly se všechny aplikace, tak jak bylo očekáváno. Dále po nahrání určité aplikace pomocí přepínače `-load` by měla jít určitá aplikace zvolit podle jména hzx souboru. To však nefungovalo a pokud bylo potřeba komunikovat s určitou aplikací, muselo se tak pomocí aid. To samé pak platí pro smazání aplikace. Při přepínači `-delete` se na kartě nic nesmazalo a musí se místo toho používat přepínač `-deleteaid` a jako parametr zadat aid aplikace, kterou bylo potřeba smazat. Některé příkazy nefungovaly, aniž by se použil parametr `-cardtype` a typ karty.

4 Praktická část

V rámci této bakalářské práce byla vytvořena aplikace s grafickým rozhraním. Aplikace dokáže vypsat informace o kartě a uložit je do souboru. Zároveň také dokáže jednotlivé funkce karty otestovat a společně i s naměřenými časy je uložit ve formátu(CSV) vhodném pro další zpracování uživatelem. Dále aplikace umožňuje spravovat aplikace na čipové kartě, tj. mazat, nahrávat nebo přehrávat. Nahrává se pomocí souboru typu `hzx`. To znamená ulehčení pro vývojáře oproti aplikaci MUtil, kde se aplikace musela ještě překládat pomocí nástroje `halugen`, což vývojáře zdržovalo od práce. Pro vývoj grafického rozhraní byla využita JavaFx, pro jádro aplikace programovací jazyk Java. Aplikace také obsahuje soubory pro bench-markové testy karet. Tyto aplikace jsou vytvořeny v programovacím jazyku C. Při testech je nahrávání na kartu plně automatické.

4.1 Výsledná aplikace na čipové kartě

4.1.1 Struktura aplikace na kartě

K volení aplikací z karty využívám mnou vytvořenou určitou hiearchii, tak aby byla možnost vše jednoduše volat případně do budoucna vyvíjet. Jedná se o soupis hexabajtových čísel, kde každé jedno značí určitou operaci. Tato čísla se pak doplňují do určitých míst v APDU. Například pro funkci DES (Data Encryption Standard) je definovaná kolonka `INS` jako 0x10, `P1` pak 0x00 značí že se má šifrovat a 0x01 dešifrovat. Takto jednotný systém funguje jak u symetrické tak asymetrické kryptografie. Zároveň pak například kódy pro symetrickou kryptografii začínají číslem jedna a pro asymetrickou číslem dvě. Toto zaručí dostatečnou přehlednost i pro další vývoj. Struktura je zobrazena v tab. 4.1, díky ní lze volat jednotlivé funkce.

4.1.2 Funkce aplikací na čipových kartách

Správa proměnných veličin

Jelikož práce s dynamickými proměnnými v operačních systémech MultOS je v podstatě nemožná, bylo potřeba si vytvořit funkce, pro odesílání a přijímaní proměnných. Tak aby bylo možné lehce nahrávat z terminálu všechny proměnné. Na kartě jsou dvě funkce a to konkrétně `setMyVariable(BYTE* data, WORD* velikost)` pro nahrávání a `getMyVariable(BYTE* data, WORD* velikost)` pro čtení z karty. První argument je odkaz na data a druhý velikost zasílaných dat. Pokud se jedná o čtení, tak pokud je ve zprávě APDU parametr `P1` roven nule, chceme kompletní celý rozsah proměnné. Tyto funkce umí pracovat až s velikostí proměnné 765 bajtů.

	INS	**P1**
0x1x - Symetrické operace		
DES	0x10	P1 ==0 pro šifrování, P1== 1 pro dešifrování
DES BLOCK CIPHER ECB	0x11	
Triple DES	0x12	
AES	0x13	
0x2x - Asymetrická kryptografie + ECC		
RSA	0x20	P1 ==0 pro šifrování, P1== 1 pro dešifrování
ECCGenerateKey	0x22	
ECDH	0x23	
ECDSA	0x24	P1 ==0 pro podpis, P1== 1 pro ověření
0x3x - Hashovací fukce		
SHA	0x31	velikost SHA funkce v bajtech
SHA1	0x32	
0x4x - Ostatní		
Generování náhodného čísla	0x40	Velikost generovaného čísla v bajtech
Test přenosové rychlosti	0x41	
0x5x - Modulární operace		
Modulární sčítání	0x50	
Modulární násobení	0x51	
Modulární mocnění	0x52	
0xBx - Numerické operace		
Sčítání	0xB0	
Násobení	0xB1	

Tab. 4.1: Definice instrukční kódu v APDU.

Tj. data se posílají přes více APDU zpráv. S tímto systémem je pak vývoj jakékoliv funkcionality mnohem rychlejší a méně náchylný na chyby.

Generování klíčů a použití RSA

Pro asymetrickou kryptografii, v tomto případě RSA potřebujeme klíče. Pro toto využívám nástroje `hkeygen` v příkazové řádce od firmy MultOS. Tento nástroj dokáže vygenerovat klíče námi zadané velikosti. Jeho největší výhodou je však, že dokáže vygenerovat kompletní soubory v jazyku C. Tyto soubory pak stačí nahrát do našeho zdrojového kódu, ve výsledné aplikaci je to vše zautomatizované a uživatel se nemusí o nic starat. Je to rychlé a efektivní řešení, jak vygenerovat potřebné klíče. O potřebnou strukturu klíčů pro operační systém MultOS se také postará `hkeygen`. Na obr. 4.1 je zobrazena ukázka generování klíčů o velikosti 2048 bitů, přepínač `-cfile` vytváří soubor v jazyku C, do příkazové řádky je však vygenerován v jiném formátu. Podporovaná velikost klíčů v operačních systémech MultOS je 2048 bitů. Při překročení definované bitové délky klíče dojde k pádu aplikace a zachycení výjimky. Uživatel je o tomto stavu informován chybovým hlášením. Velikost klíče je rovna velikosti vstupu funkce. Kryptografická struktura kryptografického páru klíčů RSA je definována v souboru `RSA.H`, ve kterém jsou i prototypy funkcí, takže jej lze lehce volat. Pro RSA šifrování se v aplikaci využívá klasické šifrování pomocí modulárního mocnění, pro dešifrování se využívá zrychlené RSA pomocí čínské věty o zbytcích.

```
C:\temp\demo-workspace\eloyalty\Debug>hkeygen -modsize 2048 -v -public MYpu
blicKey.cert -private MYprivateKey -cfile
p   = +c0f5a86f80e56d36f1e3d99799b105b994c78cad65af989b8a10be20578c545382e8c
b71ddce5c927d663fa2cbfe1a1ac985965b3975a3d3a347f396f9ce5a6527a50c9eda4b74b5
bbe8a8e65a4ddff4d3a82e5457636e83b1a0b6999224a78ac25d26196308e331eb1bf9ffb2f
473e6f04ed669306ab0eebf435937e59059d9
q   = +c4372f70cfe76fd0a081899b10427912f6a7b1d907509b29dbc5d391f47a2e812e611
b80de5c52ed9c5c9e27f33ea55f3ba7e6b6988c1eb1e482302d7f911e9ff78dba911102dbee
15feb770c9a64b95521b2c452c4d2c8656e624baf07db7745cd9c4164362a9f1e1109e39b6b
59a710ac019ecf716bfe92f6661c5801a518f
m   = +93e5ad7eb0bf977ff5e07b9791b01a814ea712b0e8007f22feb4c1dfe182091798564
cf5da9e233766fa58bea40d63885e5162222b2030d7d9f19ec765c21d506125c155245dd000
7fae055668c9537567c907b4bf8e2b7594842c44bec4dd13c66d04355bcb618f9b7257dd1e0
8f3f5f0bfbf683df31b7c76e3615376b10b9997139878489 4b44a820e2c103a23b7b8c1aece
5ba2896ac84485634b59ea0a810ca566d11b56b5bf59e1d4a309ed2809d4f9ce14b28803b85
4939f21f43ea39f636b1006d621ab0f952b059aa9fc0c339b2a5218d3f6e2c2bee710a966e7
8b33cf84f2b4457031052c43fa504697aae813e96f146392f4d725e5606e8819d937
e   = +0000c353
d   = +813a4e687f7f489e5c6eb9447342cedd36fb9ccba6fee495ba621f13d1e1b8bb88b0e
644488bd830dd45f61e4cbf2239b1411590bc6fb4cd40ce2372303cb321f9ae9ed63a817888
c8902cb084b0caf022c5a485b0b2a3faad25dbc7808d19adebc442ee55c30b07d18c3e09bb1
0396746753c9f67d391c88750c9486fb360e38822346f0403d39e5c933ce7c9a55c6b8b1427
f51c3968772e25e7ef8455f1d6df59c234aa3942f0dd8558720175948b66364677b2d240130
ccc1034cf5027df24c25547d792233d176ca5078429e4426b2775494e385c4a5db59957040a
a2efd6a9567c59197c46adde50f8b6ef0041e2531fb8b7e34bb2aee57a7dee6b092b
dp  = +4c2719123a582deb06edb7bcb005c11350a79723c377b4184c2a48188a0d17dd60db1
14afeee3f741c6fff7ed10abea0d0721b2ffa2e38f9c7f69a40c3c262546e988a88778d6221
f8490808228dc26b74a94a2d07ca083f95d4b62db59bd6039b326a7e88761373d544e377004
945ba2da388dad6d8156fad147384a148040b
dq  = +88336465cfcdb1340c12238ce9dd771dff30e9119648420d1656a39aff8081ef04234
3ba6e0b10e87be6a22d57baa44f08393c2b68901e36fb0839202e5e2f5cfb696f2f502a8ee4
bbc000f8089f333ddcc0637e03a1e061295f37ba473221b033f04c7641d72d01263260db677
595afe111d6e78caaeef70f3fd93408b5f9dd
u   = +2251496b6ffaf0c16081cca92ec4ecc6e98556149a600953c857be10957989b16fa47
441973899ae89cb1d58a9929d4165dedc9bc96cec41546d2b8148b8d9fd0e8b13a69d223a5a
110ba390022d69322146a2131670558739e2cd3110356c5022e4eaeb48e45c1d2f02d6c7a97
c156abd03e9739f0ebc543859da3a4d1f63c0
Key verification succeeded
```

Obr. 4.1: Ukázka výstupu generování klíčů pomocí hkeygen

Modul eliptických křivek

Pro výpočty s eliptickými křivkami jsou použity eliptické křivky definované standardem Brainpool [15]. Doménové parametry pro karty MultOS jsou v následujícím pořadí format, velikost, p, A, B, x, y, q, h. Kde format je stále 00 a velikost je velikost v bajtech jednotlivých elementů. A a B jsou parametry křivky, x a y jsou souřadnice generujícího bodu křivky G, q řád generujícího bodu a h - kofaktor tj. je poměr mezi řádem třídy a podgrupou[15]. Soukromý klíč se vypočítá z doménových parametrů, který je ve výsledku dlouhý od 192bitů až do 512 bitů. Rovnice eliptické křivky je zobrazena na 4.1

$$E : y^2 = x^3 + A * x + B \quad mod \quad p \tag{4.1}$$

Modulární aritmetika

Při implementaci modulárního násobení na různých kartách, lze narazit na mnoho problémů. V dokumentaci je napsáno, že délka operandů by měla být stejně dlouhá jako modulus. To však znamená, že není možné násobit řádově menší čísla než modulus. Řešením může být tzv. padding (výplň), kdy místo v bajtovém poli vyplníme nulami. Tj. výsledné číslo bude na konci pole tak jak je zobrazeno na 4.1

```
0x00000000000000000000000000000000562148593623deadbeaf
```

Výpis 4.1: Ukázka uložení na konci bajtového pole

Pokud program nahrajete na kartu ML3, může nastat problém. Karta totiž po provedení operace nahraje výsledek operace do bajtového pole úplně na začátek. Bohužel zbytek pole nevynuluje. Jelikož se výsledek ukládá do operandu 1 tak může nastat situace, kdy máme na začátku pole uložený výsledek naší operace a na konci pole pak náš vstupní operand. Ukázka výsledku operace zobrazena na 4.2

```
0xfaedbdccce25510000000000000000562148593623deadbeaf
```

Výpis 4.2: Uložení výsledku na začátku bajtového pole a vstupního operandu na konci bajtového pole

Implementace na kartě ML421 je rozdílná oproti ML3. To znamená, že jde velmi těžko vytvářet univerzální operace, které je možné nahrát na různé karty. Evidentně každý výrobce si nahrává svoje implementace základních operací. Implementace na kartě ML421 se však chová mnohem přívětivěji a to tak, že se výsledek nahraje na konec bajtového pole. Vývojář by si tedy měl před vytvořením aplikace pro více karet ověřit jejich funkčnost.

Sčítání a násobení

Při implementaci sčítání a násobení dochází dost k velkému plýtvání pamětí karty. Příkaz pro sčítání vypadá `MultOSBlockAdd ( 128 ,operand1, operand2, vysledek)`, kde první argument funkce je velikost operandů, pak dva operandy a poslední argument je adresa výsledku operace. Bohužel první argument je konstanta, což tedy znamená, že funkce není dynamická a překladač musí již v době překladu znát velikost operandů. Toto bylo vyřešeno tak, že byly vytvořeny části kódu jak je zobrazeno na 4.3 a to konkrétně od 10 do 128 bajtů.

```
if(velikostOperand1==10){
    MultOSBlockAdd ( 10 ,operand1, operand2, vysledek);
}
```

Výpis 4.3: Ukázka použití sčítání

Toto řešení je funkční, avšak každý takový řádek si zabere z paměti počet bajtů o velikosti dané operace. Jiná možnost je vytvoření funkce, která by uměla sčítat například po osmi bajtech v cyklu, což však pro bench-makrové měření není relevantní, protože funkce nebude nikdy tak rychlá jako volání funkce s předem definovanou délkou operandu.

Generování náhodných čísel

MultOS podporuje generování náhodných čísel. Jedná se o volání funkce, která po zavolání vygeneruje 8 bajtů a uloží jej na zásobník. Do celkového času je tak započten i čas pro zkopírování ze zásobníků. Jelikož nejmenší možné generování je osm bajtů, v aplikaci je ale zvoleno generování čísel od jednoho bajtu, tak výsledky jsou konstantní až do osmi bajtů.

4.1.3 Měření kryptografických operací na kartě

Implementovány jsou i různé funkce, které tam mohou být dokonce i vícekrát, takže lze porovnat jejich výkon. Například kryptografická funkce DES lze volat přímo pomocí primitiva, ale i zároveň pomocí primitiva BlockCipher, které očekává jako argument typ funkce. Takto lze například porovnat i SHA-1, kde se jedná o primitivum a zároveň se v balíku funkcí nachází SecureHash, který očekává jako argument velikost hashovací funkce.

Způsob měření

Na začátku celého měření, terminál pošle číslo v APDU zprávě, kolikrát se má provést daná operace na kartě. Čím větší číslo pošle, tím test bude přesnější, ale zároveň

bude mnohem déle trvat. Pro měření je standardně využíváno deset opakování, tím jsou dostatečně omezeny jiné vlivy. Do celkového měření se jednou započítá odeslání příkazu a jedna odpověď. Proto je dobré mít číslo větší, abychom eliminovali tyto negativní vlivy. Před samotným měřením se odešlou proměnné, které inicializují proměnné na kartě tak, aby se tento proces nezapočítával do času měření. Na počítači zároveň běží kontrola dat tak, aby bylo zajištěno, že karta vše spočítala správně a operace proběhla korektně.

Před samotným měřením si aplikace uloží startovací čas, následně proběhne volání funkce. Po skončení operace se startovací čas odečte od aktuálního času a vydělí se počtem opakování. Zobrazeno na rovnici4.2.

$$\check{c}as_{výsledný} = \frac{\check{c}as_{aktuální} - \check{c}as_{startovací}}{počet\ opakování}[\text{ms}] \tag{4.2}$$

Aplikace umí počítat i rychlost přenosu mezi terminálem a kartou, ta se počítá způsobem, který je zobrazen na rovnici 4.3

$$výsledná\ rychlost = \frac{počet\ bajtů * počet\ opakování}{\check{c}as_{aktuální} - \check{c}as_{startovací}}[\text{bajt/ms}] \tag{4.3}$$

4.1.4 Návrh struktury aplikací na kartě

Aplikace, které se využívají pro měření hodnot, jsou rozdělené do určitých oblastí a to z několika důvodů. Jeden z hlavních důvodů je přehlednost aplikací. Tím, že je v aplikaci pouze pár řádků, je kód mnohem srozumitelnější a čitelnější. Důležitějším důvodem je však to, že bylo ušetřeno místo na kartě. Některé operace si již při nahrávání ukládají na statickou paměť jako například numerické sčítání a násobení, popsáno v 4.1.2.Měření není možné pomocí jednoho souboru a to kvůli ECC (Elliptic-Curve Cryptography) výpočtům a RSA. Na kartu se totiž nahrávají klíče, velikost klíče musí být známá již při překladu, druhou variantou, by bylo mít více klíčů v jedné aplikaci, avšak toto řešení by zase znamenalo zbytečné plýtvání paměti karty. Další výhodou je modulárnost aplikací, kdy si vývojáři mohou vytvořit vlastní modul s potřebnými algoritmy pro měření a tak tuto aplikaci rozšířit.

Aplikace jsou rozděleny podle jednotlivých typu měření. Uživatel tedy může měřit symetrickou a asymetrickou kryptografii, modulární operace, operace na eliptických křivkách, numerické operace, hashovací funkce. Implementovaná je také jedna aplikace pouze k o otestovaní jednotlivých funkcí, aby programátor mohl zjistit v kratším čase, zda je funkce vůbec podporovaná.

Následující tab. 4.2 popisuje všechny použité soubory pro testování karet, které byly použity. Ve jméně těchto souborů, je vždy napsána velikost klíče a AID, které se při překladu přiřadí danému .hzx souboru.

Asymetric512_f0000011.c	Asymetrická kryptografie, s nahranými klíči 512, 1024 a 2018 bitů
Asymetric1024_f0000012.c	
Asymetric2048_f0000013.c	
Symetric_f0000010.c	Symetrická kryptografie, klíče se nahrávají dynamicky
ECC192_f0000016.c	Funkce s eliptickými křivkami, doménové parametry jsou nahrány v každém souboru zvlášť
ECC224_f0000017.c	
ECC256_f0000018.c	
ECC320_f0000019.c	
ECC384_f0000020.c	
ECC512_f0000021.c	
HashAndRng_f0000014.c	Hashovací funkce, generování náhodných čísel
Modular_f0000015.c	Modulární operace
NumericADD_f0000022.c	Numerické sčítání
NumericMUL_f0000023.c	Numerické násobení
TestClass_f0000024.c	Soubor pro základní otestování funkčnosti

Tab. 4.2: Seznam použitých C souborů pro výpočty

Při vývoji byly použity tyto hlavičkové soubory:

- `switch.h` - swich, který dokáže volat všechny funkce
- `includeVariables.h` - zde jsou definované všechny promněné a konstatny
- `ECC.h` - implementovány funkce pro práci s eliptickými křivkami
- `myRSA.h` - funkce pro práci s RSA

hlavičkové soubory `switch.h` a `includeVaribles.h` se používají ve všec C souborech. Zbylé dva jsou pouze pro konkrétní soubory. Tímto je zaručeno, že pokud se změní jedna proměnná, projeví se ve všech souborech.

4.2 PC aplikace

Aplikace byla vyvíjena v Java 8 pomocí vývojového prostředí NetBeans.Komunikace s kartou probíhá dvěma způsoby. Co se týče testování, program na přímo komunikuje s kartou a to pomocí java balíčku `javax.smartcardio`. Ten dokáže zajistit veškerou komunikaci od zahájení komunikace, výběr čtečky, výběr aplikace až po samotnou komunikaci s naší aplikací. Druhý způsob je pomocí nástroje `hterm`, který ulehčuje práci vývojáři. Má již implementované volání jednotlivých primitiv a nebylo je jej nutné znovu programovat. V některých částech je využito volání primitiv pomocí APDU příkazů. Program komunikuje s aplikací hterm pomocí příkazového řádku. Volání příkazového řádku zajišťuje volně dostupná knihovna CmdTool. S touto knihovnou lze volat jakýkoliv příkaz, který lze volat pomocí windows nástroje `cmd`.

4.2.1 Nahrávání aplikace na kartu

Záložka v aplikaci pro správu aplikací dokáže zobrazit všechny nahrané aplikace, společně s pamětí, kterou zabírá. Po výběru na danou aplikaci, je možné okamžitě aplikaci hned vymazat. To zaručuje rychlé a přehledné operace s kartou. Dále lze zde nalézt informace o celkové paměti a zbývajícím volném místu. Poslední věcí je zde nástroj pro nahrávání aplikací. Nástroj umí nahrávat soubory typu `.hzx`. Je zde možné i přehrávání, avšak nástroj `hterm` nefunguje podle očekávání a při zavolání přepínače `-reload` se aplikace nepřhraje. Aplikace si tedy projde cílový soubor, nalezne zde AID, které vymaže na kartě a poté jej na aplikaci zase nahraje. Pokud by se toto nestalo, objevila by se chybová hláška o duplikaci AID.

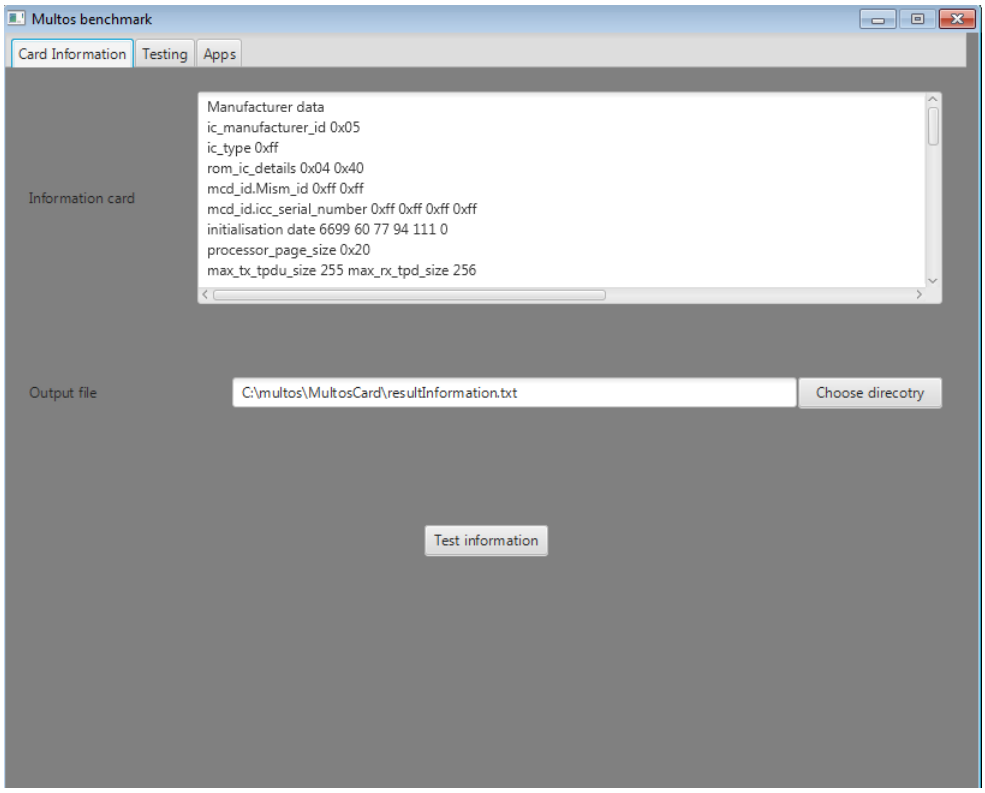
4.2.2 Struktura aplikace

V aplikaci se nachází několik tříd, které korespondují s aplikacemi na kartě, tyto třídy jsou `Asymetric`,`ECC`, `HashAndRng`, `Modular`, `NumericOperations`, `Symetric`, `TestClass`. Tyto jsou dost podobné, vždy obsahují název souborů, které nahrávají společně s AID souboru. Následně se zde vždy nachází metody pro testování daných algoritmů a kotrolu jejich správnosti. Jsou to tedy třídy, které řídí proces na kartě. Ve třídě `Variables` se nacházejí metody, pro posílání a udržování proměnných na kartě. O tom se lze více dočíst v kapitole 4.1.2. Ve třídě `CardInformation` se nacházejí všechny metody pro získávání informací o kartě, včetně zjištění jestli je karta aktivní. Dále se zde nacházejí metody pro správu aplikací. Třída `AIDS` je implementována pro udržování informací o aplikacích, které jsou nahrané na kartě. Nejhlavnější třída `CoreMultosBenchmark`, pak obsahuje metody pro zahájení komunikace, počítání času, nebo výpis a uchování informací o výsledcích měření. Pro práci s uživatelským rozhraním se využívají třídy `FXMLAPPController` a `MultosCard`. Tyto třídy načítají soubor `FXMLapp.fxml`, který definuje všechny vlastnosti elementů v uživatelském prostředí.

4.2.3 Grafické rozhraní uživatelské aplikace

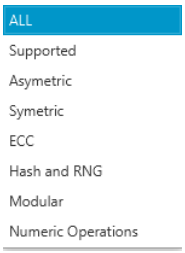
Na obr. 4.2 lze vidět, jak se aplikace zobrazí, hned po spuštění. Uživatel zde může získat informace o kartě a zároveň si jej uložit na místo, které si předem určil.

Na obr. 4.4 je zobrazena aplikace v testovacím módu. Zde uživatel má možnost zvolit, do kterého souboru se mu výsledek testování uloží. Dále pak počet opakování provedení jednotlivých operací na kartě. Čím vyšší počet opakování, tím větší přesnost výpočtu času. Další vstup od uživatele je informace o tom, co vlastně chce testovat. Má možnost, si vybrat vše, otestovat pouze kartu bez bench-markových



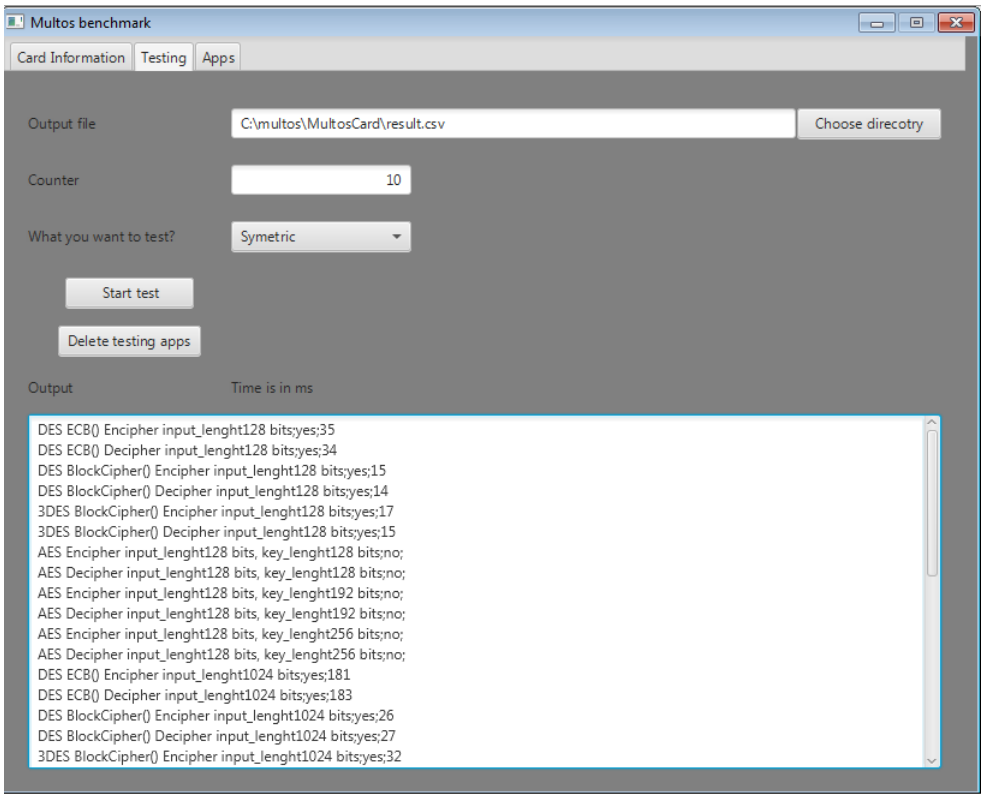
Obr. 4.2: Aplikace v záložce pro čtení informací z karty

testů a nebo jednotlivé skupiny viz obr.4.3 . Nachází se zde také tlačítko pro smazání všech aplikací, které slouží pro testování. Toto tlačítko je zde pro případ, že by nastala nějaké fatální chyba při běhu programu, tak aby bylo možné odinstalovat všechny pomocné aplikace.



Obr. 4.3: Porovnání podpisu a ověření pomocí ECDSA na kartě ML3

Na obr. 4.5 je vyobrazena poslední záložka a to pro správu aplikací na kartě. Uživatel zde po kliknutí na tlačítko aktualizovat zobrazí všechny aplikace nahrané na kartě, společně s jejich velikostmi. Následně si jej může vybrat a jednoduše smazat. Zároveň vidí kolik paměti mu zbývá. Druhou částí je část pro nahrávání aplikací, kde uživatel vybere svůj připravený hzx soubor a může jej nahrát, případně přehrát.



Obr. 4.4: Aplikace v záložce pro benchmarkové testy

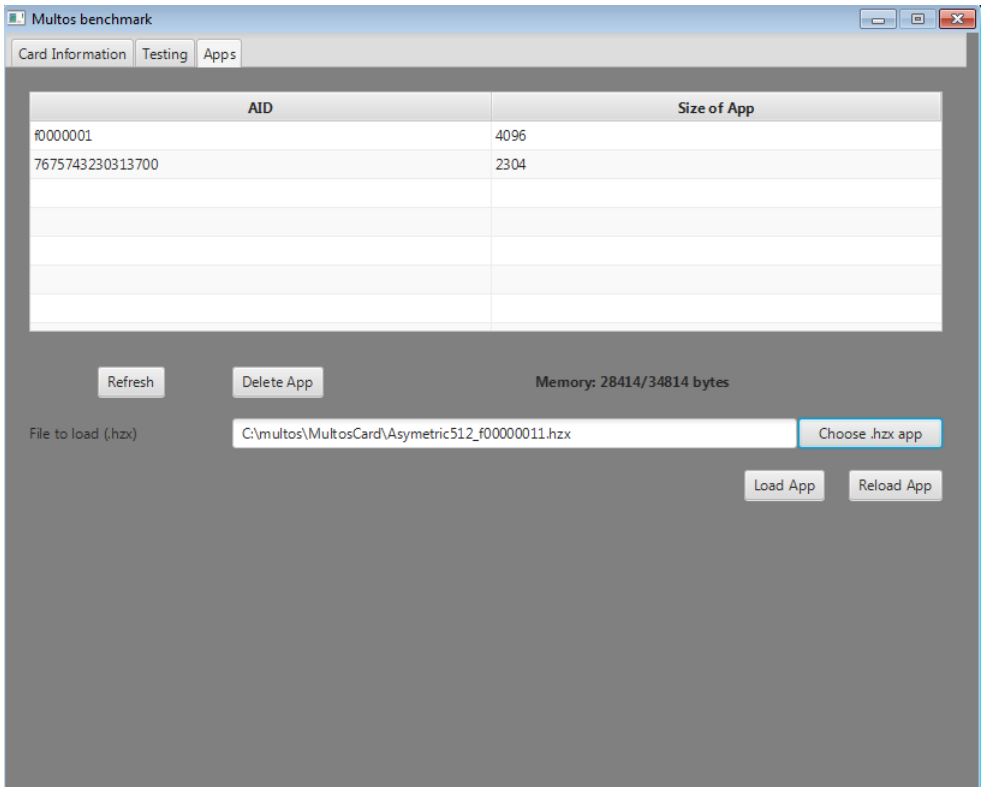
4.3 Výkonové testy kryptografických primitiv

Pro měření jsem využíval dvě karty a ty mezi sebou porovnával, viz tab. 3.1. Z výsledků je patrné, že karta ML3 je pomalejší, avšak její podpora jednotlivých funkcí je mnohem větší, tudíž je univerzálnější. Záleží na tom, co koncový uživatel bude po kartě požadovat a dle toho by si měl vybírat.

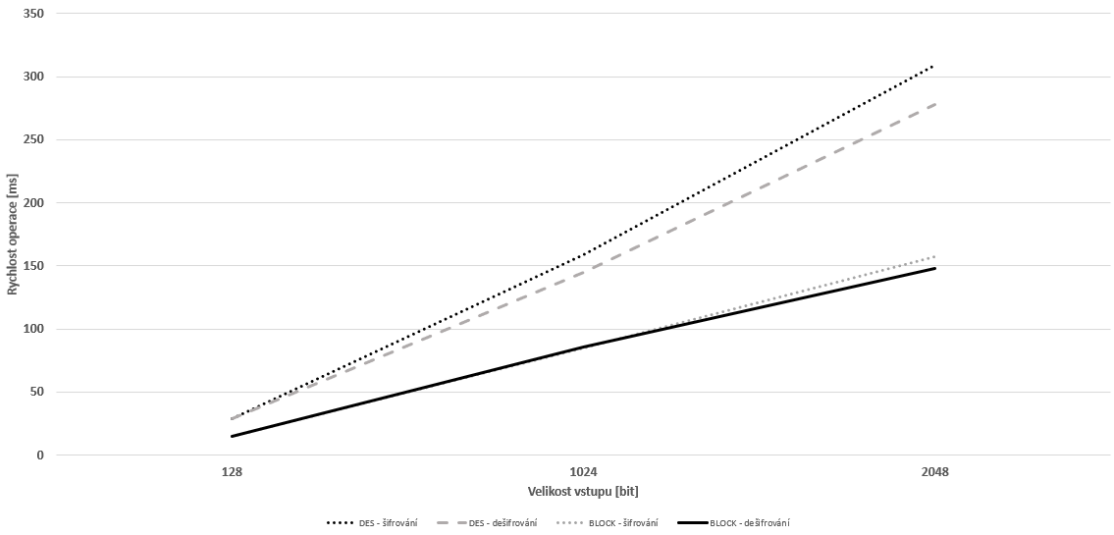
Na grafu 4.6 je vidět, porovnání různé volání funkce DES ECB pro šifrování a dešifrování a to konkrétně funkce `DES ECB Encipher \ DES ECB Decipher` a `Block encipher \ Block decpiher` pro DES ECB. Jsou zde vidět velké rozdíly v implementaci, kde blokové provedení je optimalizované a navíc mnohem jednodušší pro použití. Tato funkce zpracovává data po blocích, tudíž vidíme lineární graf závislý na délce vstupu.

Na grafu 4.7 je zobrazeno porovnání funkcí 3DES a AES se vstupním klíčem 128 bitů. 3DES je funkce jenž je lehce rychlejší o proti AES. Avšak AES je mnohem bezpečnější než 3DES a proto využití AES o proti 3DES je výhodné.

Na grafu 4.8 je vidět, že u času provedení modulárního násobení modulo 2048 bitů nejsou velké změny až do velikosti modula 512 bitů. Následně se pak časy začnou velmi zvyšovat. Nejrychlejší karta pro výpočet modulárního mocnění je ML421



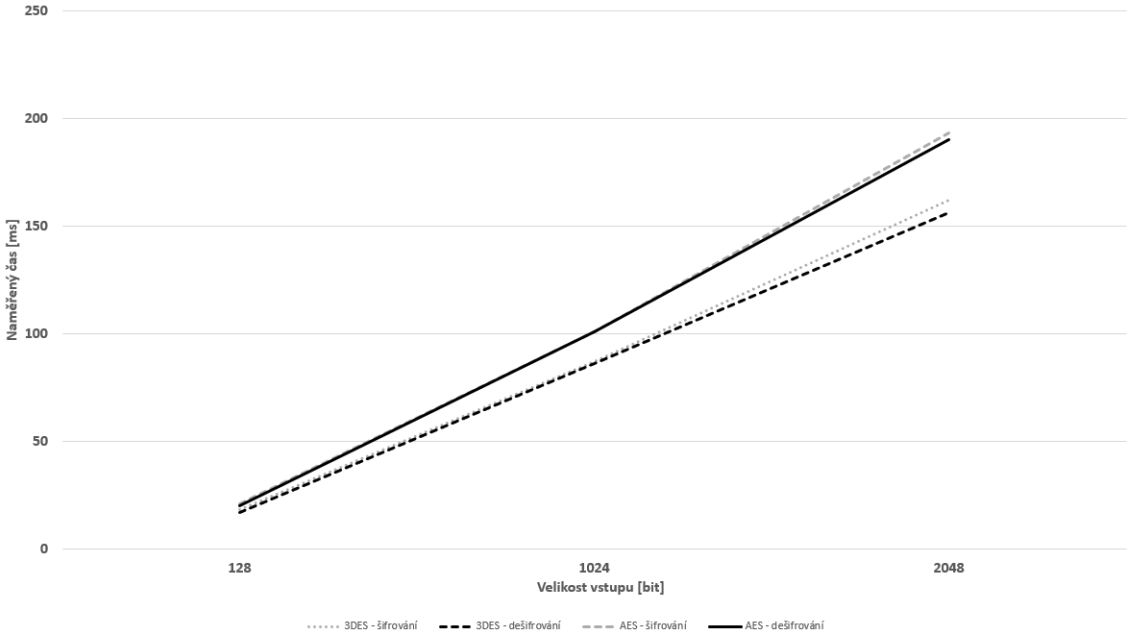
Obr. 4.5: Aplikace v záložce pro správu aplikací na kartě



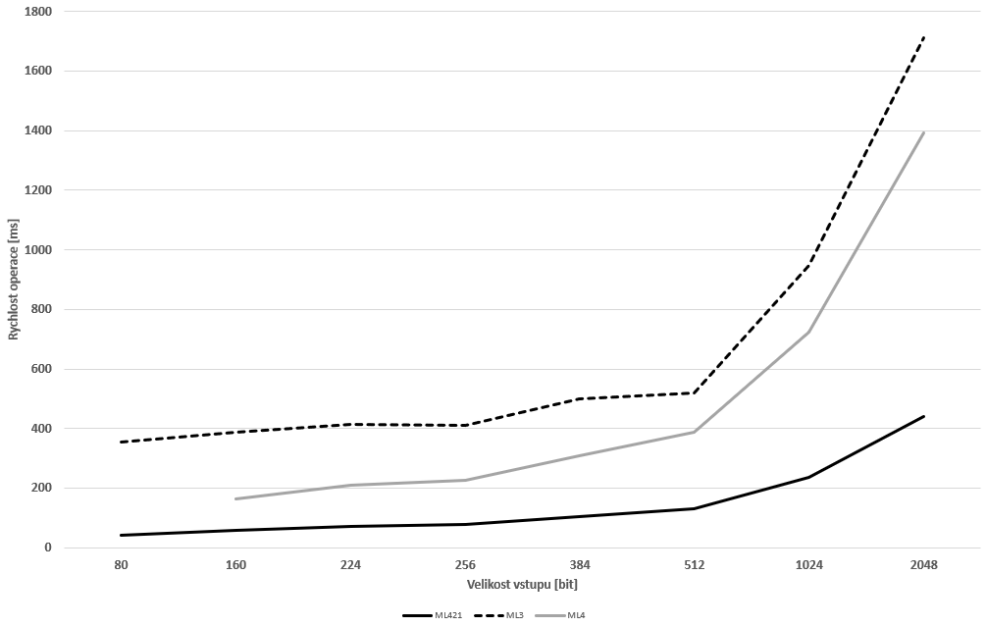
Obr. 4.6: Porovnání různých volání funkcí DES ECB na kartě ML3

následně pak ML4 a poté ML3.

Rychlosti ověření a podepisování pomocí ECDSA jsou zobrazené na grafu 4.9. Zde je znázorněno jak funkce pro ověření je časově náročnější než samotné pode-

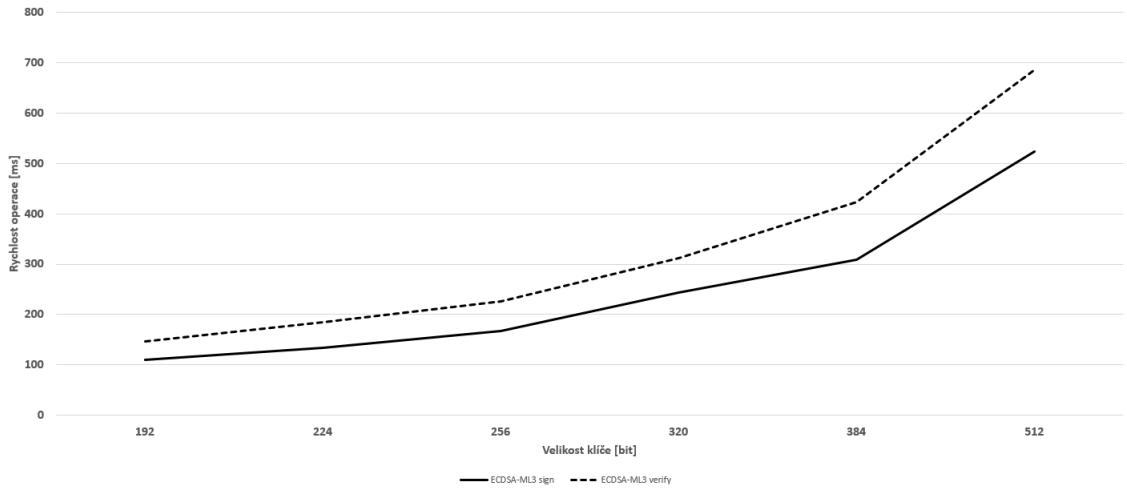


Obr. 4.7: Porovnání 3DES o proti AES se vstupním klíčem klíčem 128 bitů na kartě ML3



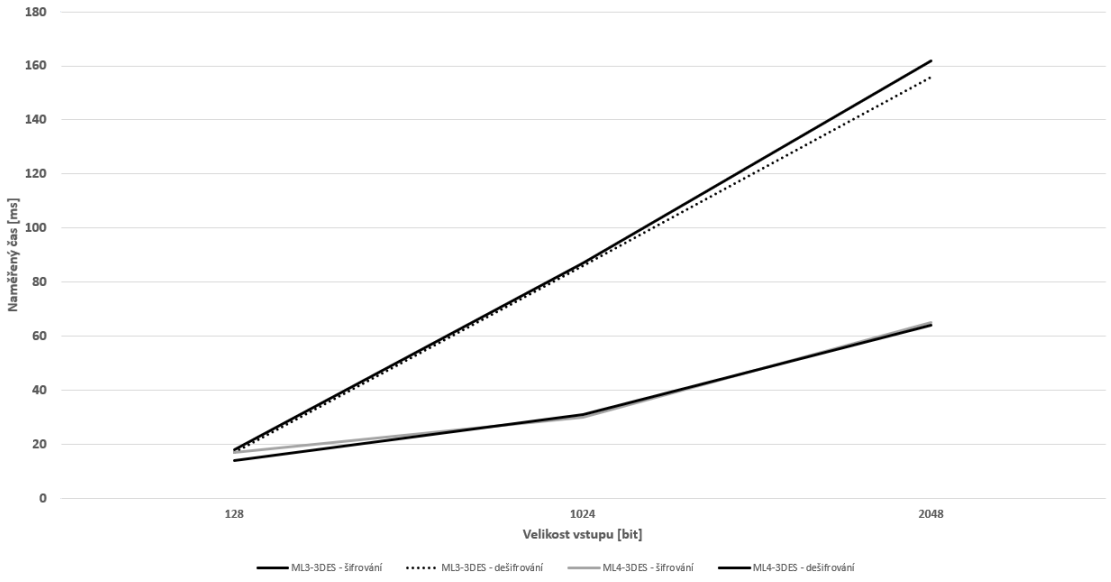
Obr. 4.8: Porovnání modulárního mocnění modulo 2048 bit

pisování. Velikost klíče je velmi závislá na rychlosti operace s tím, že čas operace roste exponenciálně. Při velikosti klíče 512 bitů je však čas stále dostatečně malý pro použití v reálné aplikaci.



Obr. 4.9: Porovnání podpisu a ověření pomocí ECDSA na kartě ML3

Graf 4.10 zobrazuje rozdíl v časech mezi výpočty 3DES mezi kartou ML3 a ML4. ML3 je mnohem pomalejší a to při nejnáročnějším výpočtu téměř o 100ms.



Obr. 4.10: Porovnání 3DES na kartách ML3 a ML4

5 Závěr

V práci byla popsána teorie čipových karet. Následně byly porovnané operační systémy a jejich kryptografická podpora. Poté je popsán samotný vývoj aplikací a různé problémy, se kterými je možno se potkat při práci s čipovými kartami s operačním systémem MultOS. Následně je popsán návrh a implementace softwarového nástroje pro testování čipových karet MultOS. V poslední části jsou shrnuty výsledky, kterých se dosáhlo.

Bylo zjištěno několik problémů, které bránily dostatečně rychlému vývoji aplikací. Největší problémem je fakt, že každá karta se může chovat trošku jinak a to velmi ztěžuje implementaci univerzální aplikace pro všechny typy karet.

Byla vytvořena aplikace, která dokáže změřit běžně dostupné funkce pro kryptografii a zaznamenat jejich výsledky na MultOS. Bylo provedeno měření asymetrických, symetrických šifrování, hashovacích funkcí, eliptických křivek, modulárních operací, generování náhodných čísel, měření rychlosti přenosu dat mezi terminálem a kartou a další funkce. Aplikace má vytvořené grafické uživatelské rozhraní, které dovoluje uživatelovi získávat informace o kartě, testovat aplikaci a spravovat aplikace, které jsou nahrané na čipové kartě. Rozhraní je jednoduché a ze začátku pomůže urychlit vývoj aplikací i začátečníkům. Správa aplikací včetně možnosti nahrávání programů na čipovou kartu je užitečná pro ladění vyvíjených aplikací. Zjištění informací o kartě je také velkou pomocí v případě, že rychle potřebujeme zjistit stav čipové karty. Testování kryptografických operací dokáže odhalit výkon karty při určitých funkcích. Aplikace je také schopna zjistit podporu kryptografických a matematických operací. Všechny získané výsledky jsou hned zobrazovány v aplikaci a zároveň ukládány do souboru.

Aplikace byla testována na třech čipových kartách s operačním systémem MultOS a každá karta byla schopna úspěšného testování.

Literatura

[1] RANKL, Wolfgang a Wolfgang EFFING *Smart card handbook. 4th ed.* Chichester: John Wiley, 2010, 1043 s. ISBN 978-0-470-74367-6

[2] HANÁČEK, Petr *Čipové karty* Computerworld [online]. 1998, [cit. 2018-10-10]. Dostupný z WWW: <http://computerworld.cz/archiv/cipove-karty-9778>

[3] Wikipedia contributors *ISO/IEC 7810* Wikipedia, The Free Encyclopedia [online]. 2019, [cit. 2018-05-05]. Dostupný z WWW: <https://en.wikipedia.org/w/index.php?title=ISO/IEC_7810&oldid=881386273>

[4] *ISO/IEC 7816-4* Identification cards — Integrated circuit cards [online]. 2019, [cit. 2018-05-05]. Dostupný z WWW: <http://www.embedx.com/pdfs/ISO_STD_7816/info_isoiec7816-4%7Bed21.0%7Den.pdf>

[5] JURGENSEN, T. M. a S. B. GUTHERY. *Smart cards* New Jersey: Prentice-Hall, 2002, 412 s. ISBN 0-13-093730-4.

[6] HAGHIRI, Y. a T. TARANTINO. *Smart card manufacturing: a practical guide.* New York: John Wiley, 2002, 221 s. ISBN 0-471-49767-3.

[7] CardLogix Corporation *CardLogic: Smart Card & Security Basics.* 2010, [cit. 2018-10-10], [online].URL http://www.smartcardbasics.com/

[8] *ISO 7816 - Smart Card Standards Overview* 2004, [cit. 2018-10-16], [online].URL http://www.smartcardsupply.com/Content/Cards/7816standard.htm

[9] *Tech-FAQ: ISO 7816.* [cit. 2018-10-15], [online]. URL http://www.tech-faq.com/iso-7816.html

[10] *SIM card format and size comparison* 2014 , [cit. 2018-10-14], [online].URL https://socialcompare.com/en/comparison/sim-card-format-and-size-comparison

[11] *Java Card Getting Started*, 2018., [cit. 2018-11-11], Oracle.com [online], URL https://www.oracle.com/technetwork/java/embedded/javacard/overview/getstarted-1970079.html

[12] *Technology Multos*, 2018., [cit. 2018-11-11], Multos.com [online], URL https://www.multos.com/technology

[13] *Java Card*The World's Leading Open Platform for Embedded Secure Elements, SIMs, and Smart Cards 2018., [cit. 2018-12-02], Oracle.com [online], URL hhttps://www.oracle.com/java/java-card.html

[14] *ZeitControl. Basiccard.*, [cit. 2018-12-02],[online], URL http://basiccard.com/

[15] D. Harkins, Ed. *RFC-6932*, Brainpool Elliptic Curves for the Internet Key Exchange (IKE) Group Description Registry ,ISSN: 2070-1721 [cit. 2018-12-02],[online], URL https://tools.ietf.org/html/rfc6932

[16] Švenda Petr *JCAlgTest*, JavaCard Algorithm Test [cit. 2018-12-10],[online], URL https://www.fi.muni.cz/ xsvenda/jcalgtest/

Seznam symbolů, veličin a zkratek

APDU	Application Protocol Data Unit
ATR	Anwer To Reset
CLA	Instruction class
DES	Data Encryption Standard
ECB	Electronic Codebook
ECC	Elliptic-Curve Cryptography
ECDH	Elliptic-Curve Diffie–Hellman
ECDSA	Elliptic Curve Digital Signature Algorithm
EEPROM	Electrically Erasable Programmable Read-Only Memory
EPROM	Erasable Programmable Read-Only Memory
INS	Instruction code
JCRE	Java Card Run-time Environment
JCVM	Java Card Virtual Machine
JVM	Java Virtual Machine
Lc	lenght command
Le	Le Field
MEL	Multos Executable Language
NPU	Numeric processing unit
OSI	Open Systems Interconnection
PPS	Protocol Parameter Selection
RAM	Random-Access-Memory
RISC	Reduced Instruction Set Computer
ROM	Read-Only-Memory
SHA	Secure Hash Algorithm
SIM	Subscriber Identity Module
SW	status word
TPDU	Transpor Protocol Data Unit
USB	Universal Serial Bus
Vcc	Supply voltage

Seznam příloh

A OBSAH PŘILOŽENÉHO CD

Složka `MutlosCard-Application`

- Spustitelná aplikace
- Testovací soubory
- Návod

Složka `MultosCard-source code`

- Zdrojové kódy aplikace pro PC
- Testovací soubory
- Návod

Složka `Sources codes-smart card`

- Zdrojové kódy aplikací pro čipové karty
- Asymetrické klíče
- Návod